Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

Tuure Savuoja

Hybridisovellukset mobiilisovelluskehityksessä

Diplomityö
Espoo, 28. huhtikuuta 2015

Valvoja: Professori Petri Vuorimaa
Ohjaaja: Diplomi-insinööri Matti Kokkola

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Tuure Savuoja		
Työn nimi:	Hybridisovellukset mobiilisovelluskehityksessä		
Päiväys:	28. huhtikuuta 2015	**Sivumäärä:**	vi + 54
Pääaine:	WWW-teknologiat	**Koodi:**	T-111
Valvoja:	Professori Petri Vuorimaa		
Ohjaaja:	Diplomi-insinööri Matti Kokkola		

Hybridisovelluksissa Web-teknologia ja käyttöjärjestelmän tarjoamat rajapinnat yhdistetään, jolloin voidaan hyödyntää kummankin teknologian tarjoamia mahdollisuuksia.

Tämä diplomityö kartoittaa asioita, jotka tulee ottaa huomioon kehitettäessä hybridisovellusta mobiililaitteille. Erityisesti tarkoituksena on löytää eroja tavallisen Web-sovelluksen toteuttamiseen. Työ on konstruktiivinen tapaustutkimus, jossa toteutetaan mobiilisovellus sekä hybridi- että Web-sovelluksena.

Hybridisovelluksen sisältämää Web-sovellusta kehitettäessä on huomioitava mobiiliympäristön erityispiirteet, ja käytettävän hybridisovelluskehyksen rajoitteet. Käyttöliittymää suunniteltaessa on varmistettava, että suunnitteluratkaisut ovat luontevia jokaisen julkaisualustan kontekstissa sekä toiminnallisesti että ulkoasullisesti. Lähes kaikkia sovelluksen osia on kuitenkin mahdollista uudelleenkäyttää hybridisovelluksen ja tavallisen Web-sovelluksen välillä.

Tässä työssä esitetyt huomiot palvelevat hybridisovelluksien ja -sovelluskehysten kehittäjiä sekä hybridisovelluksena toteutettavien mobiilisovellusten käyttäjiä.

Asiasanat:	hybridisovellus, mobiilisovellus, Web-sovellus, natiivi sovellus, uudelleenkäyttö, sovelluskauppa, sovelluksen sisäiset ostot, Cordova
Kieli:	suomi

Aalto University
School of Science

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author:	Tuure Savuoja		
Title:	Hybrid applications in mobile application development		
Date:	April 28, 2015	**Pages:**	vi + 54
Major:	WWW technologies	**Code:**	T-111
Supervisor:	Professor Petri Vuorimaa		
Advisor:	Matti Kokkola M.Sc. (Tech.)		

Hybrid mobile applications enable developers to utilize advantages of both Web technologies and native operating system interfaces.

This master's thesis aims to identify issues that should be considered in the development of hybrid applications for mobile devices. In particular, the aim is to determine the differences between the development of regular Web applications and hybrid applications. The thesis is a constructive case study, in which a mobile application is developed both as a Web application and hybrid application.

When developing a Web application contained in a hybrid application, the developer should take into account the mobile environment and the restrictions imposed by the used hybrid application framework. The user interface should feel natural in the context of all selected release platforms, regarding both visual and functional aspects. In addition, almost all components of a regular Web application can be reused in a hybrid application, and vice versa.

The findings of this thesis serve the developers of hybrid applications and hybrid application frameworks, as well as the end-users of mobile applications developed as hybrid applications.

Keywords:	hybrid application, mobile application, web application, native application, reuse, application marketplace, in-app purchases, Cordova
Language:	Finnish

Alkusanat

Haluan kiittää esimiestäni ja tämän diplomityön ohjaajaa diplomi-insinööri Matti Kokkolaa mahdollisuudesta tehdä diplomityö mielenkiintoisesta aiheesta. Kiitos koko TrademarkNow'n henkilökunnalle tuesta ja kannustuksesta. Haluan myös kiittää perhettäni ja ystäviäni. Tukenne ja pitkämielisyytenne diplomityöprojektin aikana on ollut minulle korvaamattoman arvokasta. Kiitos erityisesti vaimolleni Saaralle!

Helsingissä, 28. huhtikuuta 2015

Tuure Savuoja

Sisältö

Luku 1

Johdanto

Web (World Wide Web, WWW) on muotoutunut sovellusalustaksi, joka tarjoaa mahdollisuuden kehittää monenlaisilla Web-selaimilla, käyttöjärjestelmillä ja laitteilla toimivia sovelluksia. Nykyään puhutaankin Web-palveluiden lisäksi Web-sovelluksista. [58] Mobiilisovellukset on perinteisesti kehitetty tietylle käyttöjärjestelmälle ja laitejoukolle käyttäen käyttöjärjestelmän ja laitevalmistajan tarjoamia rajapintoja ja työkaluja. Näistä sovelluksista käytetään nimitystä natiivi sovellus. [61]

Sekä Web-sovelluksilla, että natiiveilla sovelluksilla on omat etunsa ja mahdollisuutensa. Web-sovellukset ovat luonteeltaan alustariippumattomia ja toisaalta natiivit sovelluksiet voivat hyödyntää käyttöjärjestelmänsä sovelluskauppaa ja rajapintoja [58]. Hybridisovelluksissa Web-teknologia ja käyttöjärjestelmän tarjoamat rajapinnat yhdistetään, jolloin voidaan päästä hyötymään kummankin teknologian tarjoamista mahdollisuuksista [37]. Hybridisovelluksista on tullut kiinnostava vaihtoehtoinen toteutustapa natiivien ja Web-sovellusten rinnalle.

1.1 Työn tavoitteet

Tämän työn tavoitteena on kartoittaa asioita, jotka tulee ottaa huomioon, kun kehitetään hybridisovellusta mobiililaitteille. Erityisesti tarkoituksena on

löytää eroja tavallisen Web-sovelluksen toteuttamiseen.

Tämä työ pyrkii vastaamaan seuraaviin tutkimuskysymyksiin:

TK1: Miten Web-sovellus tulee toteuttaa, jotta se on mahdollista julkaista lähes identtisenä hybridisovelluksena sovelluskaupassa?

Tätä selvitetään kuvaamalla sovelluksen toteutuksessa tehtyjä ratkaisuja, ja havaittuja ongelmia.

TK2: Kuinka laajasti sovelluksen osia voidaan uudellenkäyttää hybridisovelluksen ja Web-sovelluksen välillä?

Uudelleenkäytön laajuutta selvitetään kahdella mittarilla: Sovelluksen lähdekoodista lasketaan niiden lähdekoodirivien suhteellinen määrä, jotka on tehty tiettyä alustaversiota varten. Lisäksi toteutus- ja suunnittelutyöhön käytetyistä tunneista lasketaan tiettyä alustaversiota varten käytettyjen työtuntien suhteellinen määrä.

TK3: Onnistuuko hybridisovelluksen integroiminen yhden tai useamman sovelluskaupan sovellusten sisäisten ostojen järjestelmään?

Integroimisen onnistumista selvitetään yrittämällä integroida kehitettävä sovellus Applen AppStoren sovelluksen sisäisten ostojen järjestelmään.

Näitä asioita selvitetään konstruktiivisella tapaustutkimuksella, jossa toteutetaan mobiilisovellus sekä hybridi- että Web-sovelluksena. Konstruktiivinen tutkimusote keskittyy tosielämän ongelmien ratkaisemiseen, ja soveltuu siten hyvin työn tavoitteisiin.

Sovellukset toteutetaan TrademarkNow'lle, joka on tavaramerkkien ennakkotutkimusteknologiaa tarjoava yritys. TrademarkNow on kiinnostunut hybridisovelluksista, sillä hybridisovelluksen kehityskustannukset ovat mahdollisesti natiivia sovellusta pienemmät sovelluksen osien uudelleenkäytön myötä. Lisäksi hybridisovellus on mahdollista julkaista sovelluskaupassa, jolloin

on mahdollista hyödyntää sovelluskaupan sovellusten sisäisten ostojen järjestelmää.

Tässä työssä käsitellään sovellusten kehittämistä vain kuluttajille suunnatuille mobiililaitteille. Tällaisia laitteita ovat taulutietokoneet, älypuhelimet ja näihin verrattavat laitteet. Tässä työssä ei käsitellä sellaisia alaustariippumattomaan sovelluskehitykseen tarjolla olevia työkaluja, joissa sovellusta ei rakenneta Web-teknologialla.

1.2 Työn rakenne

Luku 2 käsittelee natiivien mobiilisovellusten kehittämistä ja jakelua yleisimmille mobiililaitteiden käyttöjärjestelmille. Luvussa käydään läpi yleisimmät mobiililaitteiden käyttöjärjestelmät sekä niiden sovelluskaupat ja sovelluskehitystyökalut. Luku 3 esittelee Webiä sovellusalustana, ja käy läpi Web-sovellusten kehittämistä. Luku 4 tarkastelee hybridisovelluksen määritelmää ja rakennetta, ja esittelee hybridisovellusten kehittämiseen tarjolla olevia sovelluskehyksiä. Luvussa 5 käydään läpi sovellus, joka toteutettiin tämän diplomityön käytännön osuutena. Luvussa 6 analysoidaan sovelluksen toteutusta eri näkökulmista ja esitetään analyysin tulokset. Luku 7 tarkastelee työtä ja sen tuloksia laajemmassa mittakaavassa, ja esitettää vastaukset tutkimuskysymyksiin.

Luku 2

Mobiililaitteiden käyttöjärjestelmät ja sovellukset

Tämä luku käsittelee natiivien mobiilisovellusten kehittämistä ja jakelua yleisimmille mobiililaitteiden käyttöjärjestelmille. Aluksi käydään läpi erilaisia mobiililaitteita, ja tarkastellaan käyttöjärjestelmien ja sovelluskauppojen merkitystä loppukäyttäjän ja sovelluskehittäjän näkökulmista. Tämän jälkeen esitellään tarkemmin muutamia yleisimpiä käyttöjärjestelmiä.

2.1 Mobiililaitteet

Erilaisia kuluttajille suunnattuja mobiililaitteita on valtavasti. Moninaisuudesta huolimatta tietyt asiat pätevät lähes kaikkiin laitteisiin ja yhteisiä ominaisuuksia on löydettävissä.

Mobiililaitteita voidaan luokitella esimerkiksi näytön fyysisen koon, laitteen ominaisuuksien, hinnan tai käyttöjärjestelmän mukaan. Rajanvedot näissä luokitteluissa ovat subjektiivisia ja häilyviä, mutta erilaisten käsitteiden käyttäminen on silti hyödyllistä tarkasteltaessa mobiililaitteita ja -sovelluksia.

Näytön fyysisen koon perusteella laitteet voidaan asettaa janalle, jonka ääripäinä ovat pienempikokoisemmat puhelimet (mobile phones) ja suurempi-

kokoisemmat tabletit eli taulutietokoneet (tablets). Kuvassa 2.1 on typillisen kokoinen puhelin ja tabletti. Vielä puhelimiakin pienemmät puettavat älylaitteet (wearables), kuten älykellot (smart watches), ovat myös nousemassa uudeksi tuoteryhmäksi puhelimien ja tablettien rinnalle [32].

Puhelimet jaetaan usein vielä useampaan eri ryhmään, sillä ne ovat myyntimäärällä mitattuna selkeästi suosituin tuoteryhmä [31]. Ominaisuuksien perustella puhelimet voidaan jakaa peruspuhelimiin (feature phones) ja älypuhelimiin (smartphones).

Peruspuhelimet edustavat älypuhelimia edeltävää aikakautta, jolloin kaikki sovellukset (ominaisuudet) tulivat yleensä puhelimen mukana. Mikäli sovellusten asentaminen oli mahdollista, se ei ollut nopeaa ja yksinkertaista. Älypuhelimissa sen sijaan sovellukset ovat keskeisessä roolissa, ja niitä voi asentaa helposti puhelimen mukana tulevan käyttöjärjestelmän sovelluskaupasta. Lähes kaikista älypuhelimista ja tableteista löytyy kosketusnäyttö ja langaton matkapuhelinverkkoa käyttävä internet-yhteys.

Kuva 2.1: Älypuhelin ja tabletti.

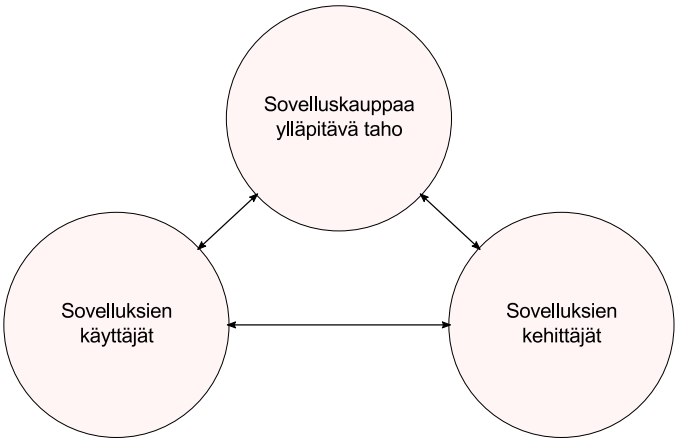
2.2 Käyttöjärjestelmän merkitys

Jokainen mobiililaite tarvitsee käyttöjärjestelmän, joka toimii alustana, jonka päällä natiivi sovellus voi toimia. Käyttöjärjestelmä on keskeisessä roolissa sovelluksiin perustuvissa älylaitteissa, kuten älypuhelimissa ja tableteissa, niin loppukäyttäjän kuin sovelluskehittäjänkin näkökulmasta. Tähän on pohjimmiltaan kaksi toisiinsa liittyvää teknistaloudellista syytä. Ensinnäkin tietylle käyttöjärjestelmälle kehitetty natiivi sovellus toimii yleensä vain kyseisessä käyttöjärjestelmässä, ja sen kehittäminen vaatii kyseisen käyttöjärjestelmän työkalujen, ohjelmointirajapintojen (Application Programming Interface, API) ja käytäntöjen tuntemusta. Toisekseen, edellisestä syystä johtuen, saman sovelluksen tekeminen natiivina sovelluksena erikseen usealle käyttöjärjestelmälle vaatii kehittäjältä moninkertaisen työmäärän [37].

Käyttöjärjestelmän merkitystä vahvistaa mobiilisovellusmarkkinoiden kehitys, jossa sovelluskaupoista on tullut sovellusten ensisijainen välityskanava kehittäjiltä loppukäyttäjille. Loppukäyttäjät haluavat luonnollisesti valita käyttöjärjestelmän, jolle on saatavilla paras mahdollinen valikoima käyttäjälle hyödyllisiä ja mieleisiä sovelluksia. Sovelluskehittäjät taas ovat kiinnostuneita käyttäjämäärältään suurimmista käyttöjärjestelmistä. Näin loppukäyttäjien ja sovelluskehittäjien tarpeet kohtaavat muodostaen kaksipuoliset markkinat, joiden alustana toimii käyttöjärjestelmän sovelluskauppa. [39]

2.3 Sovelluskaupat

Sovelluskauppa yhdistää loppukäyttäjät, sovelluksien kehittäjät ja sovelluskauppaa ylläpitävän tahon muodostaen ympärilleen sovellusekosysteemin [43] (kuva 2.2). Sovelluskauppa tai -kaupat ovat osa käyttöjärjestelmän arvolupausta loppukäyttäjälle, ja siten merkittävä käyttöjärjestelmän kilpailutekijä. Käyttöjärjestelmän kehittäjä ylläpitää usein myös sovelluskauppaa ja pyrkii sitä kautta tukemaan pääasiallista liiketoimintaansa. Applen tapauksessa sovellustarjonta tukee laitteiden myyntiä. Google taas pyrkii saamaan lisää



Kuva 2.2: Sovelluskaupan muodostama ekosysteemi.

käyttäjiä mainosverkolleen. [41] Toinen motivaattori sovelluskaupan ylläpitäjille ovat myyntitulot [41]. Sovelluskaupan ylläpitäjä ottaa itselleen osan sovelluskaupan myynnistä, vaikka suurin osa myyntituloista välitetäänkin kehittäjille [39].

Sovelluskaupan ylläpitäjä pyrkii edistämään kaksipuolisen markkinan toimimista ja omaa hyötyään tukemalla sovelluskehittäjiä ja käyttäjiä. Monet käyttäjärjestelmien kehittäjät pyrkivät parantamaan sovelluskauppansa sovellusvalikoimaa tarjoamalla sovelluskehittäjille ilmaisia kehitystyökaluja ja kirjastoja. Käyttäjiä taas ohjataan sovelluskaupan käyttäjiksi integroimalla käyttöjärjestelmän kehittäjän oma sovelluskauppa osaksi käyttöjärjestelmää. [39]

Sovelluskehittäjille sovelluskauppa tarjoaa markkinointi- ja myyntikanavan, jonka kautta voi yleensä hoitaa myös sovelluksen maksuliikenteen. Sovelluskauppassa on yleensä mahdollista julkaista sekä ilmaisia, että maksullisia sovelluksia [43]. Lisäksi monien sovelluskauppojen kautta on mahdollista välittää sovelluksen sisäisiä ostoja. Sovelluskaupasta riippuen sovelluksen sisäisillä ostoilla voi tarjota esimerkiksi sovelluksen tarjoaman palvelun määräaikaista tilausta tai muita virtuaalisia kulutushyödykkeitä. [42, 43]

2.4 Käyttöjärjestelmän natiivit rajapinnat

Sovelluskaupan lisäksi käyttöjärjestelmä tarjoaa ohjelmointirajapinnat laitteen ja käyttöjärjestelmän ominaisuuksiin, kuten esimerkiksi laitteen kameraan tai käyttöjärjestelmän ilmoituskeskukseen. Käyttöjärjestelmän tarjoamista ohjelmointirajapinnoista käytetään nimitystä natiivit ohjelmointirajapinnat. Ne tarjoavat raamit, joiden avulla on mahdollista rakentaa käyttöjärjestelmän käytäntöjen mukaisia, ja muiden sovellusten kanssa yhteentoimivia sovelluksia. Yhtenäinen käyttöliittymä ja käyttökokemus sovellusten välillä parantaa käyttäjäkokemusta. Tämän vuoksi käyttöjärjestelmillä on omat käyttöliittymäohjeet, joissa käyttöjärjestelmän käyttöliittymäkäytännöt on esitetty. Eri käyttöjärjestelmien käytännöt eroavat toisistaan. Kehitettäessä sovellusta useammalle alustalle onkin tasapainoteltava sovelluksen eri alustaversioiden yhtenäisyyden ja alustan eri sovellusten yhtenäisyyden välillä. [45]

Natiivien ohjelmointirajapintojen käyttäminen mahdollistaa yleensä edelleen parhaan mahdollisen suorituskyvyn, joskin esimerkiksi Web-teknologioiden suorituskyky on kehittynyt. Tämä on tärkeää erityisesti mobiililaitteissa, joissa akun varuksen kesto on tärkeää maksimoida. Pelit ja muu interaktiivinen grafiikka vaativat usein suorituskykynäkökulmien erityistä huomioimista. [20]

2.5 Käyttöjärjestelmien markkinaosuudet

Tällä hetkellä yleisimmät mobiililaitteiden käyttöjärjestelmät ovat Googlen Android ja Applen iOS. Gartnerin mukaan vuonna 2013 myydyistä älypuhelimista 94 %:ssa oli Android tai iOS. [29] Taulutietokoneissa tilanne on samankaltainen, sillä lähes 98 %:ssa oli jompikumpi näistä kahdesta käyttöjärjestelmästä [30]. Muista käyttöjärjestelmistä merkittävät osuudet ovat Microsoftin Windows-käyttöjärjestelmän eri mobiililaiteversioilla sekä BlackBerryn samannimisellä käyttöjärjestelmällä. Seuraavissa luvuissa esitellään

tarkemmin näitä yleisimpiä käyttöjärjestelmiä.

2.6 Android

Android on Googlen avoimen lähdekoodin käyttöjärjestelmä, jonka kehitystyössä on mukana monia tahoja. Kuka tahansa saa käyttää ja muokata käyttöjärjestelmää vapaasti, mutta virallisen Google Play -sovelluskaupan saavat käyttöönsä vain laitteet, jotka Google on tarkastanut yhteensopiviksi. [34] Virallisen Google Play -kaupan lisäksi Android-sovelluksia tarjoaa kuitenkin myös useita pienempiä kolmannen osapuolen sovelluskauppoja, kuten Amazon Appstore ja SlideMe [43]. Google Play -sovelluskauppa tarjoaa sovelluksen sisäisten ostojen järjestelmää, jossa voi myydä sekä kertalaskutteisia tuotteita (managed in-app products) sekä tilauksia (subscription), joille voi valita erilaisia laskutusvälivaihtoehtoja [35].

Monet laitevalmistajat käyttävät Androidia ja se on käytössä hyvin monenlaisissa laitteissa. Laaja laitevalikoima antaa käyttäjille vaihtoehtoja, mutta saattaa myös aiheuttaa sovelluskehittäjille lisätyötä yhteensopivuusongelmien muodossa. [53] Älypuhelinten ja tablettien lisäksi Androidia hyödynnetään nykyään myös televisioissa ja autoissa.

Androidin virallinen kehitysympäristö Android Studio on vapaasti saatavilla Linux, Mac OS X ja Windows -käyttöjärjestelmille [33]. Sovellusten ohjelmointi tapahtuu Java-kielellä.

2.7 iOS

iOS on Applen mobiililaitteiden, kuten iPhone-älypuhelimen ja iPad-tabletin, käyttöjärjestelmä. [7] Se on suljettua lähdekoodia ja saatavilla vain Applen omiin laitteisiin valmiiksi asennettuna. Sovelluskauppoja on vain yksi, Apple App Store, eikä sovellusten asentaminen laitteisiin muualta ole mahdollista. App Store tarjoaa sovelluksen sisäisten ostojen järjestelmää virtuaalisten kertakulutus- ja kestohyödykkeiden (consumable, non-consumable) sekä

kesto- ja määräaikaistilauksien (auto-renewable subscriptions, non-renewable subscriptions) myymiseen [6].

Kuka tahansa voi kehittää sovelluksia ilmaisen rekisteröitymisen jälkeen, mutta sovellusten asentaminen laitteeseen ja julkaiseminen App Store -sovelluskauppaan vaatii vuosimaksullisen jäsenyyden iOS-sovelluskehittäjäohjelmassa (iOS Developer Program) [9]. Sovellusten kehittäminen vaatii Applen Mac-tietokoneen, sillä tarvittava Xcode-kehitysympäristö on saatavilla vain Mac OS X -käyttöjärjestelmälle. iOS-sovellusten pääasiallinen ohjelmointikieli on Objective-C [8].

2.8 Muut käyttöjärjestelmät

Windows on Microsoftin käyttöjärjestelmä, josta on saatavilla lukuisia eri versioita erilaisiin laitteisiin. Windows on suljettua lähdekoodia, mutta sitä käyttäviä laitevalmistajia on useita. Laitteesta ja käyttöjärjestelmän versiosta riippuen sovelluskauppana toimii Windows Store tai Windows Phone Store. Sovellusten julkaiseminen näissä sovelluskaupoissa vaatii vuosimaksullisen jäsenyyden [49]. Sovellusten kehittämiseen on saatavilla ilmainen Visual Studio Express -kehitysympäristö. Käytettävän ohjelmointikielen voi valita muutamasta eri vaihtoehdosta. Valittavana on esimerkiksi C♯ ja JavaScript. [50]

BlackBerry on samannimisen yrityksen kehittämä mobiililaitteiden käyttöjärjestelmä, joka on saatavilla vain kyseisen yrityksen mobiililaitteiden mukana. Sovelluksia voi kehittää esimerkiksi Javalla, ja julkaista BlackBerry World -sovelluskaupassa. [16]

Luku 3

Web sovellusalustana

Tämä luku esittelee Webiä sovellusalustana. Aluksi käydään Web-sovellusten kehittämisen kannalta keskeisimmät tekniikat, ja tarkastellaan Web-sovellusten arkkitehtuuria. Tämä jälkeen käsitellään tekijöitä, jotka mahdollistavat saman Web-sovelluksen ajamisen erilaisissa laitteissa.

3.1 Perustekniikat

Web hyödyntää vahvasti asiakas–palvelin-arkkitehtuuria. Palvelimilla on saatavilla resursseja (tiedostoja), joiden tunnisteena toimii URL-osoite (Uniform Resource Locator) [15]. Resurssit voivat olla mitä tahansa dataa, esimerkiksi Web-sivuja tai kuvia. Asiakas, kuten Web-selain, hakee käyttäjän antaman URL:n osoittaman resurssin ja esittää sen käyttäjälle. Tiedonsiirtoon asiakkaan ja palvelimen välillä käytetään yleensä HTTP-protokollaa, joka on tekstipohjainen pyyntö–vastaus-protokolla. [24]

Kolme Web-sisällön luomiseen käytettävää perustekniikkaa ovat HTML (HyperText Markup Language), CSS (Cascading Style Sheets) ja JavaScript. Web-sivun sisältö ja rakenne kuvataan HTML:ää käyttäen. Sisällön esitystapa, kuten tekstin tyylit ja värit, määritellään CSS:n avulla. JavaScript-ohjelmat voivat käyttää selaimessa olevia rajapintoja, jolloin Web-sivusta saadaan toiminnallinen ja interaktiivinen Web-sovellus. Yksi tärkeimmistä ra-

japinnoista on DOM (Document Object Model), jonka avulla ladatun sivun sisältöä on mahdollista muokata paikallisesti [40]. Web-sivut sisältävät usein viittauksia muihin resursseihin, kuten kuviin, toisiin HTML-sivuihin, CSS-tyyleihin ja JavaScript-ohjelmiin.

Perustekniikoiden kehittyminen on tehnyt Webistä sovellusalustan. Sovellusten kannalta keskeistä on ollut etenkin JavaScript-ohjelmointirajapintojen kehitys. Yksi ensimmäisistä merkittävistä edistysaskeleista oli Ajax (Asynchronous JavaScript and XML). [58] Ajax on joukko tekniikoita, joiden avulla osa Web-sivun sisällöstä voidaan päivittää ilman, että koko sivua tarvitsee ladata palvelimelta uudestaan [28]. Tärkein osa Ajaxia on XHR-ohjelmointirajapinta (XMLHttpRequest), joka mahdollistaa HTTP-pyyntöjen tekemisen sivunlatauksen jälkeen [60].

Web-sivu voi normaalisti tehdä Ajax-pyyntöjä vain samalta palvelimelta peräisin oleviin resursseihin. Tämä rajoitus perustuu saman alkuperän käytäntöön (same-origin policy). [13]. CORS (Cross-Origin Resource Sharing) mahdollistaa hallitut poikkeamat saman alkuperän käytännöstä, ja siten HTTP-pyyntöjen tekemisen eri alkuperää olevien resurssien välillä [59].

3.2 HTML5 ja uudet tekniikat

Ajaxin jälkeen ohjelmointirajapintojen valikoima on monipuolistunut merkittävästi. HTML:n uusin versio HTML5[14] on yksi merkittävistä uusista standardeista. Se tarjoaa monia uusia ominaisuuksia, joista on hyötyä erityisesti Web-sovellusten toteuttamisessa. [58]

Termiä HTML5 käytetään myös yleisesti viittaamaan laajempaan joukkoon Web-tekniikoiden uusia versioita, joista kaikki eivät sisälly varsinaiseen HTML5-standardiin [2]. Tällaisia tekniikoita ovat esimerkiksi Geolocation API[1], WebGL[2] ja uudet CSS standardit[3].

[1] `http://www.w3.org/TR/geolocation-API/`
[2] `https://www.khronos.org/registry/webgl/specs/1.0/`
[3] `http://www.w3.org/Style/CSS/specs`

Web-selaimissa saatavilla olevien standardoitujen rajapintojen määrä kasvaa jatkuvasti. Kehitteillä olevat uudet rajapinnat tekevät Webistä entistäkin paremman sovellusalustan. Esimerkiksi Web Notifications tarjoaa mahdollisuuden näyttää ilmoituksia käyttöjärjestelmän ilmoituskeskuksessa [36], ja Service Workers monipuolistaa Web-sovelluksien toimintamahdollisuuksia yhteydettömässä tilassa [54].

Myös tiedonsiirtoon on kehitetty uusia tekniikoita HTTP:n rinnalle, kuten WebSocket ja WebRTC. Nämä laajentavat perinteistä pyyntö–vastaus-mallia tarjoamalla mahdollisuuden reaaliaikaiseen tiedonsiirtoon asiakkaan ja palvelimen välillä. WebRTC tukee lisäksi vertaisverkkoyhteyksiä kahden selaimen välillä. Myös HTTP-protokollasta on kehitteillä uusi versio, joka pyrkii parantamaan suorituskykyä.

3.3 Web-sovelluksen arkkitehtuuri

Web-selain toimii Web-sovellukselle alustana samalla tavalla kuin käyttöjärjestelmä toimii natiiville sovellukselle, kuten Web-selaimelle. Joissain käyttöjärjestelmissä Web-selain on upotettu osaksi käyttöjärjestelmää niin, että kaikki sovellukset ja käyttöjärjestelmän kuori (shell) on toteutettu Web-sovelluksina. [3, 57] Tällaiset Web-selainpohjaiset käyttöjärjestelmät eivät kuitenkaan vielä ole saavuttaneet merkittävää käyttäjäkuntaa.

Web-sovellus on selaimessa pyörivä JavaScript-sovellus, joka hyödyntää Web-selaimen tarjoamia JavaScript-ohjelmointirajapintoja ja muita Web-tekniikoita. Tietojen hakemista ja päivittämistä varten sovellus tekee taustalla Ajax-pyyntöjä palvelimille. Ajaxia hyödyntävät Web-sovellukset ovat usein yhden sivun sovelluksia (single-page application, SPA). Tällöin käyttöliittymänä on HTML:n ja CSS:n avulla toteutettu Web-sivu, johon kulloinkin esitettävä sisältö päivitetään JavaScriptin avulla. Tämä eroaa perinteisistä monen sivun sovelluksista (multi-page application, MPA), jossa sovellus koostuu tarpeen mukaan palvelimelta noudettavista kokonaista Web-sivuista [48]. Samassa sovelluksessa on mahdollista yhdistellä kumpaakin mallia,

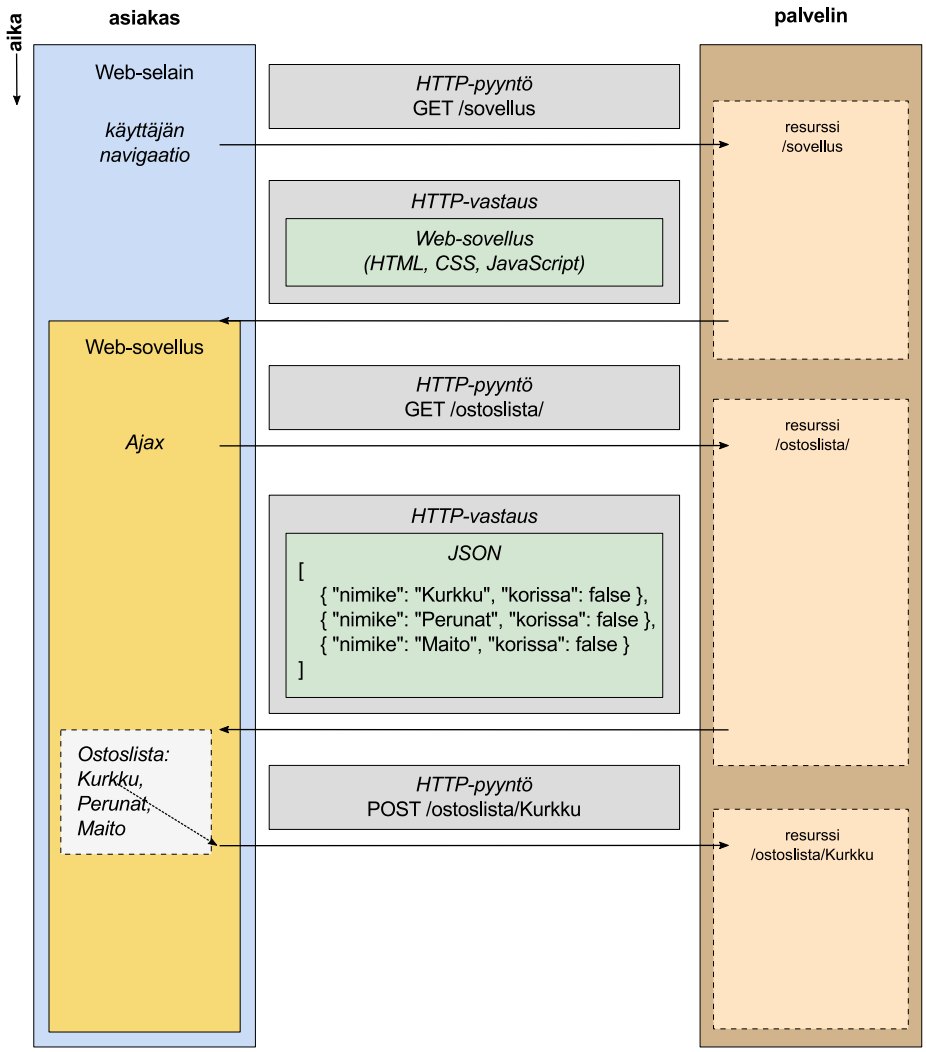
mutta nykyään monet sovellukset ovat täysin yhden sivun sovelluksia.

Yhden sivun sovelluksien Ajax-pyynnöissä siirrettävä tieto on usein esitetty valmiin HTML-esitysmuodon sijaan jossain rakenteisessa muodossa. Usein käytettyjä muotoja ovat esimerkiksi JSON (JavaScript Object Notation) [17], joka on yksinkertainen tekstipohjainen tiedonvälitykseen kehitetty tiedostomuoto, ja enemmän ominaisuuksia tarjoava XML (Extensible Markup Language) [18]. Sovelluksen käyttämä palvelin toimiikin sovelluksen tiedostojen jakamisen lisäksi ohjelmointirajapintana, jota sovellus hyödyntää. Tämä palvelimen tarjoama ohjelmointirajapinta suunnitellaan usein REST-tyylin (Representational State Transfer) mukaisesti, jolloin sovelluksen mallintamat kohteet esitetään HTTP-resursseina [25]. Kuvassa 3.1 on esitetty yksinkertaistettu esimerkki tällaisesta Web-sovelluksesta.

3.4 Työpöydältä mobiiliin

Webin alkuperäinen käyttötarkoitus oli staattisten dokumenttien jakaminen. Se on kuitenkin kehittynyt monien vaiheiden kautta palvelu- ja sovellusalustaksi. Web-teknologian kehittymisen myötä Web-sovellukset voivat tarjota työpöytäsovellusten kaltaisia toiminnallisuuksia. [58]

Myös laitteet, joilla Webiä käytetään ovat muuttuneet. Applen iPhone toimi tienraivaajana Webin mobiilikäytölle tarjoamalla käyttökokemuksen, joka oli merkittävästi samanlainen kuin Webin käyttö työpöytäkoneella. Samat Web-sivut ja -palvelut, joihin käyttäjät olivat tottuneet, olivat nyt käytettävissä myös mobiililaitteella. [62]. Nykyään noin kolmasosa kaikesta käytöstä tapahtuu mobiililaitteilla [55]. Jotta sama Web-sovellus olisi käytettävissä jokaisessa laitteessa, täytyy kiinnittää huomiota käytettävyyteen, teknisen yhteensopivuuteen ja sovelluksen jakeluun.



Kuva 3.1: Yksinkertaistettu esimerkki REST-rajapintaa hyödyntävästä yhden sivun Web-sovelluksesta.

3.5 Sama sovellus jokaisessa laitteessa

Kun Webin mobiilikäyttö alkoi kasvaa, osa Web-palveluista alkoi tarjota erillistä mobiilikäyttöön optimoitua sivustoa [19, 52]. Erilaisten mobiililaitteiden kirjo on kuitenkin entisestään monipuolistunut, ja Web-palvelun tulisi olla käytettävä kaiken kokoisilla näytöillä. Mobiililaitteiden tehot ovat myös kasvaneet, joten ne jaksavat pyörittää monimutkaisempia Web-sivuja. Näistä syistä responsiivinen suunnittelu, jossa sama sivusto mukautuu laitteen ominaisuuksien mukaan [47], on kasvattanut suosiotaan. [51] Mobiililaitteilla on kuitenkin edelleen ominaispiirteitä, jotka on huomioitava. Esimerkiksi koskетusnäytöllisissä käyttöliittymissä ei ole vastinetta hiiren kursorin viemiselle käyttöliittymäelementin päälle (hover).

Web-sovellukset toimivat käyttöjärjestelmästä riippumatta, mutta vastaavasti käytettävän Web-selaimen pitää tukea kaikkia käytettyjä tekniikoita, jotta sovellus toimii. Monet Web-selaimet ovat saatavilla useille käyttöjärjestelmille. Lisäksi Web-tekniikoiden standardointi mahdollistaa teoriassa sen, että sama sovellus toimii eri selaimilla. Käytännössä eri selainten välillä on kuitenkin yhteensopivuusongelmia. [46]

3.6 Jakelu ja maksaminen

Web-sovellusten asennus ja jakeluprosessi on loppukäyttäjän näkökulmasta kevyt, sillä Web-sovelluksia ei tarvitse erikseen asentaa tai päivittää. Kun käyttäjä navigoi sovelluksen URL-osoitteeseen, Web-selain lataa puuttuvat ja päivittyneet sovelluksen osat palvelimelta. [3] Web-sovellusten jakelu ei ole keskitettyä, vaan se tapahtuu jokaisen sovelluksen omilta palvelimilta.

Webissä ei ole keskitettyä maksujärjestelmää esimerkiksi sovellusten sisäisiä ostoja varten, mutta vaihtoehtoja maksujen välittämiseen on tarjolla runsaasti. Keskitetyn maksuratkaisun puute on haaste käyttäjäkokemuksen kannalta, mutta sovelluskehittäjälle se tarjoaa tavan välttää maksamasta sovelluskaupan ylläpitäjälle osuutta myynnistä.

Luku 4

Hybridisovellukset

Tämä luku käsittelee hybridisovellusten kehittämistä. Aluksi käydään läpi mitä hybridisovelluksella tarkoitetaan. Tämän jälkeen käsitellään hybridisovellusten arkkitehtuuria ja kehittämistä. Lopuksi esitellään hybridisovellusten kehittämiseen tarjolla olevia sovelluskehyksiä.

4.1 Hybridisovellus käsitteenä

Käsitteelle hybridisovellus ei ole kirjallisuudessa yhtä vakiintunutta määritelmää. Yleisesti sillä kuitenkin tarkoitetaan sovellusta joka hyödyntää käyttöliittymässään sekä Web-teknologioita, että tietyn käyttöjärjestelmän natiiveja sovelluskehitystyökaluja ja -kirjastoja, ja jolla on tästä syystä natiivia sovellusta vähemmän riippuvuuksia tiettyyn alustaan [21, 37, 44].

Tiettävästi sanaa hybridi käytti ensimmäisen kerran tässä merkityksessä Lee Barney kahdessa toukokuussa 2008 julkaistussa blogikirjoituksessa. Ensimmäisessä kirjoituksessa hän käsittelee Applen iOS-sovelluskehitystyökalujen senhetkisen testiversion uutta UIWebView-komponenttia, joka mahdollistaa Web-selainnäkymän upottamisen natiiviin iOS-sovellukseen [12]. Toisessa kirjoituksessa hän puolestaan esittelee kehittämäänsä QuickConnect nimistä sovelluskehystä, joka hyödyntää kyseistä komponenttia. Kirjoituksessaan Barney käyttää tästä sovelluskehyksestä jo nimeä hybridisovelluskehys (hy-

brid framework) [11].

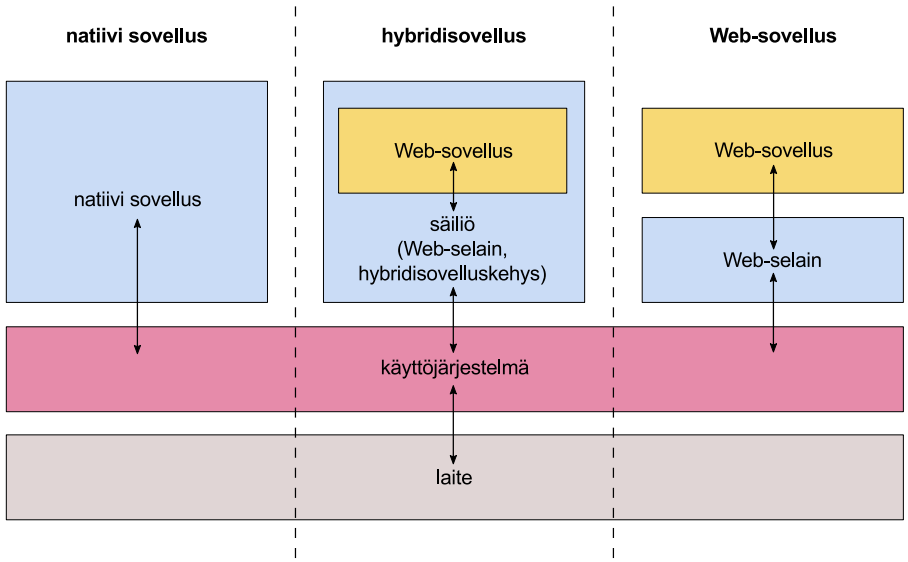
Tässä työssä termillä hybridisovellus tarkoitetaan sovellusta, joka on rakennettu seuraavanlaisesti: Natiivin sovelluksen käyttöliittymä ja mahdollisesti myös muut osat koostuuvat osittain tai kokonaan Web-sovelluksesta, jota ajetaan natiiviin sovellukseen upotetussa Web-selaimessa. Web-selain on upotettu natiiviin sovellukseen, niin että Web-sovellus on loppukäyttäjän näkökulmasta saumaton osa kokonaisuutta. Tämä määritelmä rajaa hybridisovellusten ulkopuolelle esimerkiksi sovellukset, joissa Web-teknologioita käytetään vain data siirtämiseen, ja käyttöliittymä on toteutettu natiivina. Tällaisia sovelluksia ei yleisesti pidetä hybridisovelluksina.

4.2 Arkkitehtuuri

Hybridisovellukseen upotetusta Web-selaimesta käytetään myös nimitystä säiliö (container), sillä se tarjoaa ympäristön jossa Web-sovellus voi toimia. Samoin kuin tavalliseenkin Web-selaimeen, säiliöön on mahdollista kirjoittaa laajennoksia (plugin), joiden avulla on mahdollista luoda uusia rajapintoja natiivin sovelluksen ja Web-sovelluksen välille. Käytännössä uusien rajapintojen luominen tarkoittaa, että JavaScript-ohjelmasta on mahdollista kutsua selaimen normaalien JavaScript-ohjelmointirajapintojen lisäksi omia natiivin ohjelmakoodin puolella määriteltyjä funktioita. Koska natiivin sovelluksen puolelta on edelleen pääsy käyttöjärjestelmän ohjelmointirajapintoihin, on tällä tavoin mahdollista käyttää käyttöjärjestelmän ohjelmointirajapintoja Web-sovelluksessa. [44]

Hybridisovelluksen arkkitehtuuri on kerroksinen, mutta samalla se muodostaa yhteen paketoidun kokonaisuuden. Tätä rakennetta on havainnolistettu kuvassa 4.1, jossa on esitetty rinnakkain natiivin sovelluksen, hybridisovelluksen ja Web-sovelluksen perusarkkitehtuuri.

Hybridisovellusten alustariippumattomuus saavutetaan käyttämällä sovelluskehystä, joka toteuttaa tarjoamansa ohjelmointirajapinnat jokaiselle kohdealustalle hyödyntäen kohdealustan natiiveja rajapintoja. Tärkeimpiä so-



Kuva 4.1: Natiivin sovelluksen, hybridisovelluksen ja Web-sovelluksen perusarkkitehtuuri.

velluskehyksen hyödyntämiä rajapintoja ovat kunkin alustan Web-näkymäkäyttöliittymäkomponentti (Web view), joka mahdollistaa säiliössä sijaitsevan Web-sivun esittämisen osana natiivin sovelluksen näkymää. [37]

4.3 Käyttöliittymä

Säiliönä toimivan Web-selaimen normaalit käyttöliittymäelementit, kuten osoiterivi ja valikot, on poistettu. Kun hybridisovelluksen käynnistää, sen sisältämä Web-sovellus aukeaa koko näyttöalueen kokoisena, kuten natiivit sovellukset. Hybridisovellus ei myöskään näy välilehtenä käyttäjän normaalisti käyttämän Web-selaimen puolella, vaan kyseessä on itsenäinen ja erillinen selaininstanssi. Sovelluksen vaihtaminen hybridisovelluksesta toiseen ajossa olevaan sovellukseen tai takaisin toimii aivan kuten natiivien sovellusten kohdalla. Tarkoituksena on, että hybridisovelluksen käyttäjä ei huomaa käyttävänsä Web-selaimessa pyörivää Web-sovellusta, vaan että käyttökokemus on saumaton ja vastaa natiivia sovellusta.

Jotkut sovelluskehykset tarjoavat mahdollisuuden sijoittaa sekä natiiveja käyttöliitymäkomponentteja että Web-näkymiä samaan sovellukseen. Natiiveja käyttöliittymäkomponentteja ei kuitenkaan ole mahdollista sijoittaa Web-näkymän sisään vaan natiiveilla käyttöliittymäkomponenteilla vaihdetaan kulloinkin näkyvissä olevaa Web-näkymää. Mikäli natiiveja käyttöliittymäkomponentteja ei käytetä, on erityisesti vaarana että alustan käyttöliittymäohjeet unohdetaan, sillä sovelluksen alustaversioiden yhtenäisyys on eräänlainen lähtötilanne. Tästä syystä Hybridisovelluksien käyttöliittymä on usein epäyhtenäinen alustan muiden sovellusten kanssa [37].

4.4 Sovelluskehitys

Sovelluskehittäjän kannalta hybridisovelluksen kehittäminen tapahtuu muutamia lisävaiheita lukuunottamatta kuten normaalin Web-sovelluksen kehittäminen. Hybridisovellus koostetaan jokaista alustaa varten Web-sovellukses-

ta paketoimalla se sovelluskehyksen kanssa natiiviksi sovelluspaketiksi [37]. Sovelluskehyksestä riippuen koostaminen tehdään paikallisesti tai pilvipalvelussa. Mikäli koostaminen tehdään paikallisesti, jokaisen kohdealustan omat sovelluskehitystyökalut tulee olla käytettävissä. Kehittämisen aikana koostettua sovellusta voidaan ajaa oikeassa laitteessa tai virtuaalilaitteessa, kuten iOS-simulaattorissa tai Android-emulaattorissa. Sovelluksesta riippuen pelkkää Web-sovellusta voi myös olla mahdollista ajaa sellaisenaan tavallisessa Web-selaimessa. Valmis sovellus julkaistaan lopuksi halutuissa sovelluskaupoissa.

Hybridisovellus on samaan aikaan käyttäjän ja käyttöjärjestelmän näkökulmasta natiivi sovellus, mutta kuitenkin sovelluskehittäjän näkökulmasta Web-sovellus. Kehittäjän näkökulmasta hybridisovellus pääsee hyötymään monista Web-sovelluksien eduista, kuten alustariippumattomuudesta, sillä samaa Web-sovellusta voidaan ajaa selaimesta ja käyttöjärjestelmästä riippumatta. Tästä syystä hybridisovellusten kehityskustannukset ovat pienemmät kuin natiivin kun julkaistaan useammalle alustalle [37]. Samalla hybridisovelluksilla on kuitenkin pääsy natiiveihin ohjelmointirajapintoihin ja se voidaan julkaista sovelluskauppaan ja asentaa sieltä laitteeseen. Näin voidaan natiivien sovellusten tavoin hyötyä sovelluskaupasta myynti- ja markkinointikanavana. Hybridisovellukset vastaavat tällä tavoin Web-sovellusten rajoitteisiin mobiililaiteympäristössä, silti säilyttäen osan Web-sovellusten eduista.

4.5 Sovelluskehykset

Tunnetuin hybridisovelluskehys on avoimen lähdekoodin Apache Cordova, joka tunnettiin aikaisemmin nimellä PhoneGap. Cordovaa käytettäessä sovellus kehitetään tavallisen Web-sovelluksen tapaan käyttäen HTML-, CSS- ja JavaScript-tekniikoita. Cordova tarjoaa JavaScript-ohjelmointirajapintoja, joiden kautta on mahdollista käyttää käyttöjärjestelmän natiiveja rajapintoja. Valmiita laajennoksia saatavilla runsaasti, ja monet niistä ovat myös avointa lähdekoodia. [26]

Cordovan pohjalle on rakennettu monia lisäominaisuuksia tarjoavia sovelluskehyksiä kuten uusi kaupallinen Adobe PhoneGap ja AppGyver Steroids. PhoneGapin tarjoamia lisäominaisuuksia ovat pilvessä toimiva käännöspalvelu, nopeampi tapa päivittää sovellus laitteeseen kehityksen aikana, sekä yrityksille suunnatut tukipalvelut. [56] Myös Steroids tarjoaa vastaavanlaisia palveluita, mutta siinä paikallinen koostaminen ei ole lainkaan mahdollista. Steroids tarjoaa lisäksi mahdollisuuden hyödyntää natiiveja käyttöliittymäkomponentteja siten että ohjelmassa on useampi WebView joiden välillä natiivit komponentit vaihtavat. Steroidsin myös mainitaan tukevan http-skeeman mukaisten URL-osoitteiden käyttöä. Steroids tukee vain iOS ja Android alustoja. Android-emulaattorin käyttöä ei tueta. [5]

Monet alustariippumattomien mobiilisovellusten kehitykseen tarjolla olevat sovelluskehykset eivät hyödynnä Web-teknologioita täysimittaisesti eivätkä siten ole aiemmin esitetyn määritelmän mukaan hybridisovelluksia. Esimerkiksi Appcelerator Titanium -sovelluskehyksessä ohjelmointikieli on JavaScript, mutta normaalien Web-selainrajapintojen sijaan sovellus rakennetaan Titaniumin omien rajapintojen päälle, ja lopputulos käännetään natiiviksi sovellukseksi. [4]

Luku 5

Mobiilisovelluksen toteutus

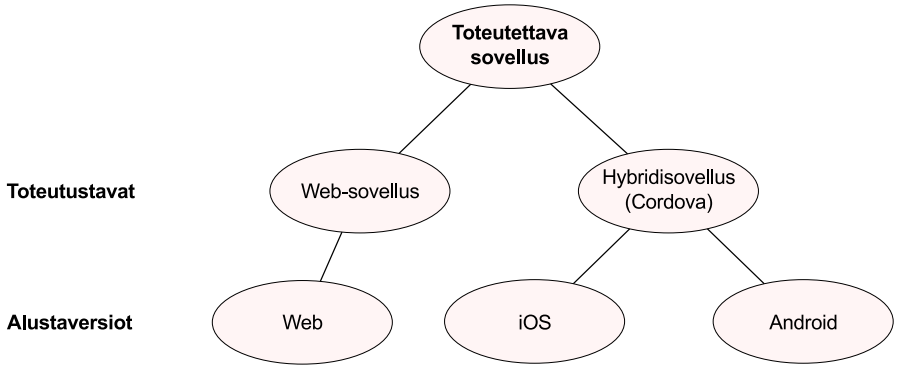
Tässä luvussa käydään läpi sovellus, joka toteutettiin tämän diplomityön käytännön osuutena. Aluksi tarkastellaan sovelluksen lähtökohtia. Tämän jälkeen esitetään sovelluksen arkkitehtuuri, ja käydään läpi toteutuksessa käytettäviä tekniikoita.

5.1 Sovelluksen tarkoitus ja vaatimukset

Kehitettävä sovellus on käyttöliittymä TrademarkNow'n uuteen, vielä julkaisemattomaan, tavaramerkkihakupalveluun. Koska palvelua ei ole vielä julkaistu, sovelluksen konkreettista toimintaa on mahdollista esitellä vain pintapuolisesti. Sovelluksen keskeisin toiminnallisuus on tekstipohjaisten hakujen tekeminen. Käyttäjä syöttää haluamansa hakuparametrit, jonka jälkeen sovellus lähettää ne TrademarkNow'n palvelimelle. Sovellus saa palvelimelta hakutulokset, jotka esitetään käyttäjälle.

Seuraavaksi käydään läpi lähtökohtia ja sovellukselle asetettuja vaatimuksia, jotka tunnistettiin ennen sunnittelu- ja kehitystyön aloittamista tai sen alkuvaiheessa.

Palvelun kohderyhmä on kansainvälinen, joten sovelluksen käyttöliittymän on oltava helppokäyttöinen ja käännettävissä useille kielille. Sovelluksen en-



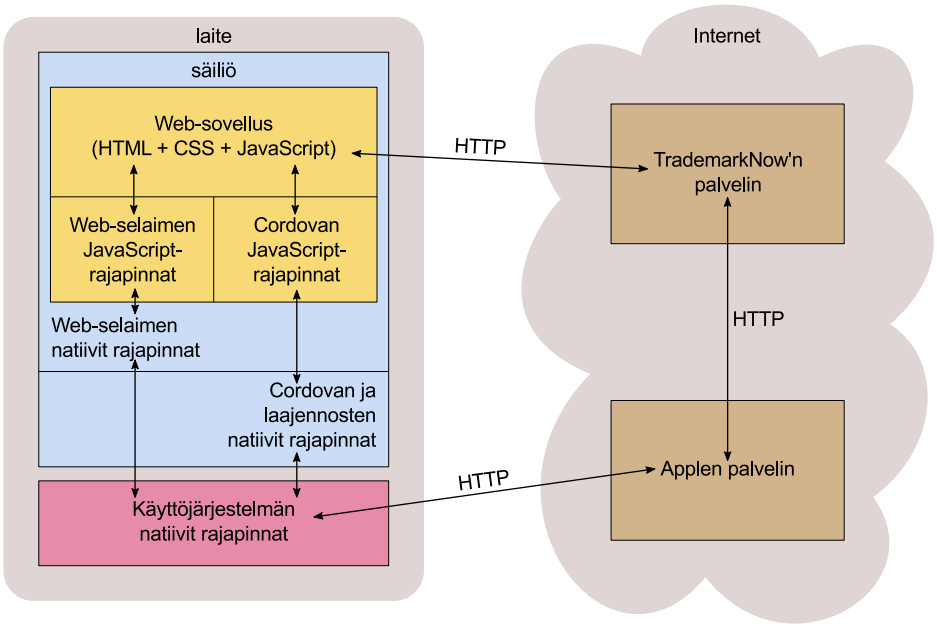
Kuva 5.1: Kehitettävä sovellus toteutetaan sekä Web-sovelluksena, että hybridisovelluksena.

sisijainen kohdelaiteryhmä on tabletit, mutta sovelluksen tulisi toimia myös pienemmillä ja suuremmilla näytöillä.

Sovelluksen toteutuksessa halutaan uudelleenkäyttää muiden TrademarkNow'n sovelluksien olemassaolevaa HTML-, CSS- ja Javascript-lähdekoodia. Valmis sovellus tullaan julkaisemaan ainakin yhdessä sovelluskaupassa, jolloin voitaisiin hyödyntää mahdollisuuksien mukaan sovelluksen sisäisiä ostoja.

5.2 Julkaisu ja testaus

Liiketoiminnallisista syistä sovellus julkaistaan ensimmäisenä iOS-alustalle. Tästä syystä sovelluksen sisäiset ostot toteutetaan toistaiseksi vain iOS-alustaa varten. Sovellus tullaan mahdollisesti julkaisemaan myöhemmin Androidille ja Web-sovelluksena. Sovelluskaupassa julkaisemista varten sovellukselle on tuotettava kuvake, ruutukaappauskuvat sekä esittelytekstit ja -kuvat. Nämä toteutetaan vasta sovelluksen ensimmäisen julkaisuversion ja palvelun lopullisen brändi-ilmeen valmistuttua. Kun sovelluksesta julkaistaan sovelluskauppaan uusi versio, sovelluksen asentaneet saavat siitä sovelluskaupan kautta ilmoituksen. Sovelluksen todennäköiset julkaisualustat ja alustaversioiden toteutustavat on esitetty kuvassa 5.1.



Kuva 5.2: Sovelluksen arkkitehtuuri (iOS-alustaversio)

Kehityksen aikana sovellusta ajetaan oikeassa iOS-laitteessa, sekä lisäksi ajoittain Web-sovelluksena tavallisessa työpöytäselaimessa, Android-emulaattorissa ja iOS-simulaattorissa. Näin pyritään takaamaan sovelluksen toimivuus erilaisissa ajoympäristöissä. Koska sovelluksen sisäiset ostot ovat tiivisti kytköksissä julkaisualustaan, ne tulevat toimimaan vain oikeassa iOS-laitteessa. Kehitystyönaikaisessa käytössä painotetaan näistä syistä iOS-tablettia.

5.3 Arkkitehtuuri

Sovelluksen iOS-alustaversion arkkitehtuuri on esitetty kuvassa 5.2. Web-alustaversion arkkitehtuuri on muuten samanlainen, mutta se ei käytä sovelluskaupan palvelinta. Arkkitehtuuriltaan sovellus on yhden sivun sovellus, joka tekee Ajaxilla REST-kutsuja TrademarkNow'n REST-ohjelmointirajapintana toimivalle palvelimelle. Tiedonsiirrossa sovelluksen ja palvelimen välillä

käytetään pääasiassa JSON-tiedostomuotoa. Sovellus käyttää Web-selaimen ja Cordovan rajapintoja, joiden kautta tapahtuu myös kommunikointi sovelluskaupan palvelimen kanssa. Myös TrademarkNow'n palvelin kommunikoi sovelluskaupan palvelimen kanssa sovelluksen sisäisten ostojen varmentamiseksi.

Koska sovelluksen Web-alustaversiota tullaan todennäköisesti jakelemaan eri verkkotunnuksesta kuin missä TrademarkNow'n REST-ohjelmointirajapintapalvelin sijaitsee, käytetään sovelluksessa CORSia. REST-kutsujen vastauksiin lisätään CORSia varten otsakeet `Access-Control-Allow-Origin` ja `Access-Control-Allow-Credentials`. Ensimmäisen otsakkeen arvo on verkkotunnus, jossa sovellus sijaitsee. Toisen otsakkeen arvo on `true`, joka tarkoittaa, että kutsun mukana voidaan lähettää eväste (cookie). Lisäksi XHR-ohjelmointirajapintaa käytettäessä pitää asettaa `withCredentials`-valitsin päälle, jotta eväste lähetetään Ajax kutsujen mukana. Hybridisovellusta varten Cordovan asetuksiin pitää asettaa REST-ohjelmointirajapintapalvelimen verkkotunnus sallittujen etäkoneiden listalle.

Lähdekoodin uudelleenkäytön helpottamiseksi käyttöliittymäelementit ja muut sovelluksen osat rakennetaan uudelleenkäytettäviksi komponenteiksi, joilla on selkeät riippuvuussuhteet. Käyttöliittymästä pyritään tekemään mahdollisimman responsiivinen, jotta se olisi käytettävä mahdollisimman monenlaisilla laitteilla.

5.4 Käytettävät sovelluskehykset ja kirjastot

Käytettäväksi hybridisovelluskehykseksi haluttiin valita jokin tunnettu ja vakaa ratkaisu. Siksi päätettiin valita Apache Cordova tai jokin siihen pohjautuvia hybridisovelluskehys. Cordovaan pohjautuvista sovelluskehyksistä tarkempaan vertailuun otettiin mukaan lupaavilta vaikuttaneet AppGyver Steroids ja Adobe PhoneGap.

Käytettäväksi hybridisovelluskehykseksi valittiin Apache Cordova. AppGyver Steroidsin tarjoamat natiivit käyttöliittymäkomponentit ovat kiinnostavia,

mutta niiden käyttäminen olisi rajoittanut sovelluksen toteutustavan monen sivun sovellukseksi. Steroidsin pilvessä toimiva käännöspalvelu olisi myös ollut ulkoinen riippuvuus, jollaista ei haluttu. Steroidsin dokumentaatio ei tuntunut olevan ajan tasalla, mikä herätti epäluottamusta. Adobe PhoneGapin tarjoamista lisäominaisuuksista ei arveltu olevan hyötyä.

TrademarkNow'n muiden sovellusten olemassaolevat käyttöliittymäkomponentit on toteutettu pääasiassa React-kirjastoa käyttäen. Jotta myös näitä komponentteja olisi helppo uudelleenkäyttää, sovelluksen käyttöliittymä toteutetaan React-komponentteina. React luo käyttöliittymän selaimessa oman DOM-ohjelmointirajapintatoteutuksensa avulla, ja käyttää tämän jälkeen selaimen DOM-ohjelmointirajapintaa lopputuloksen näyttämiseen. Tämä mahdollistaa erinomaisen suorityskyvyn [22], joka on tärkeää paljon hakutulostietoja näyttävän sovelluksen tapauksessa.

Listauksessa 5.1 on esitetty esimerkki käyttöliittymäkomponentista, joka esittää sille annetun datan kahdessa eri välilehdessä. Käyttäliittymäkomponentti koostuu React-komponentista ja siihen liittyvistä tyyleistä. Tyylit on kirjoitettu Stylus-kielellä, joka käännetää CSS-lähdekoodiksi [1]. Esitetty käyttöliittymäkomponentti hyödyntää muita komponentteja selkeästi määriteltyjen riippuvuuksien mukaisesti.

Muita sovelluksen toteutuksessa käytettäviä kirjastoja on esitetty taulukossa 5.1. Moment.js[1], Polyglot.js[2] ja Select2[3] tarjoavat toiminnallisuuksia, joita Web-selainten JavaScript-ohjelmointirajapinnoissa ei vielä ole tarjolla. Osa kirjastoista, kuten jQuery[4] ja lodash[5], tarjoavat olemassaoleviin toiminnallisuuksiin sovelluskehittäjän kannalta miellyttävämmän ohjelmointirajapinnan.

[1] `http://momentjs.com/`
[2] `http://airbnb.github.io/polyglot.js/`
[3] `https://select2.github.io/`
[4] `https://jquery.com/`
[5] `https://lodash.com/`

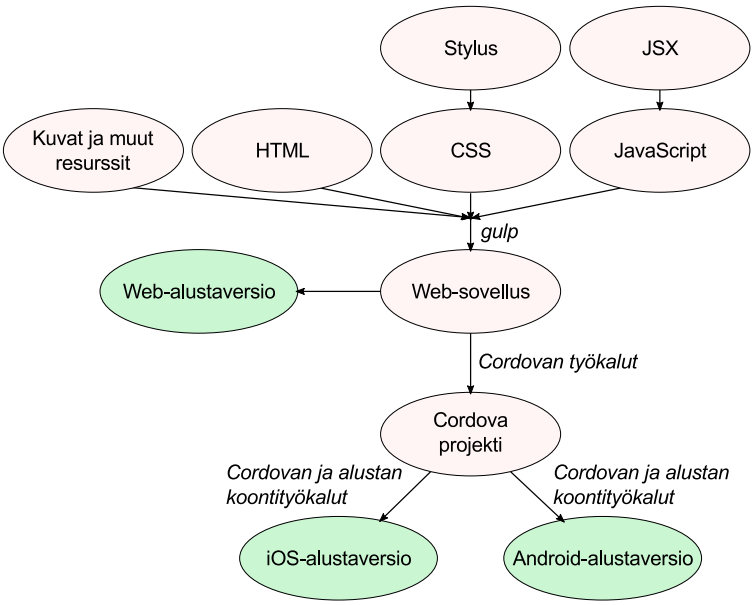
Nimi	Käyttötarkoitus
jQuery	Mukavampi ohjelmointirajapinta DOM-operaatioihin ja Ajax-pyyntöjen tekemiseen.
lodash	Työkaluja JavaScriptin arvojen ja kokoelmien käsittelyyn.
Moment.js	Ajan ja päivämäärien esittäminen ja tulkinta.
Polyglot.js	Käyttöliittymän tekstien näyttäminen valitusta käännöstiedostosta.
React	Käyttöliittymän rakententaminen uudelleenkäytettävistä komponenteista.
Select2	Alasvetovalikko, josta on mahdollista hakea valittavia arvoja.

Taulukko 5.1: Sovelluksen käyttämiä kirjastoja.

5.5 Koonti ja eri versiot

Sovelluksen koontiprosessi on kolmivaiheinen (kuva 5.3). Ensimmäisessä vaiheessa sovelluksen lähdekoodia muokataan eri tavoin halutunlaisen lopputuloksen saamiseksi ja kehitystyön nopeuttamiseksi. Toisessa vaiheessa rakennetaan Cordova-projekti Web-sovelluksen ympärille Cordovan työkalujen avulla. Kolmannessa vaiheessa kootaan alustakohtainen natiivi sovelluspaketti Web-sovelluksesta ja Cordovasta käyttäen Cordovan ja alustan koontityökaluja.

Koontiprosessin ensimmäinen vaihe on automatisoitu gulpin avulla, joka on JavaScript-pohjainen koontiautomaatiotyökalu. Web-sovelluksen koonnissa hyödynnetään useita työkaluja, joita on listattu taulukossa 5.2. Ensimmäisessä vaiheessa esimerkiksi Reactin käyttämä JSX-notaatio käännetään JavaScript-lähdekoodiksi [23], ja Stylus-kielellä kirjoitetut tyylit CSS-lähdekoodiksi. CSS-



Kuva 5.3: Sovelluksen koontiprosessin päävaiheet.

lähdekoodia muokataan vielä tämän jälkeen käyttäen Rework-kirjastoa[6]. Sovelluksen JavaScript-lähdekoodi tarkastetaan ESLint-työkalulla[7]. Se tarkistaa, että lähdekoodi on hyvien käytäntöjen ja tyylin mukainen, yrittää löytää mahdollisia virheitä. Tarkastetut ja muokatut JavaScript- ja CSS-lähdekoodit yhdistetään yksittäisiksi tiedostoiksi, jolloin tarvittavien HTTP-kyselyiden määrä vähenee merkittävästi. JavaScript-lähdekodin yhdistämisessä käytetään Browserifyä[8]. Lopuksi HTML-, CSS- ja JavaScript-lähdekoodit sekä muut sovelluksen resursseit kootaan yhteen hakemistoon Web-sovellukseksi.

Koontivaiheessa valitaan kehitys- ja tuotantokoontiversioiden väliltä. Kehitysversiossa on mukana lisätarkistuksia sekä muita kehitystyötä helpottavia tietoja, jotka kasvattavat kootun ohjelmiston kokoa. Tuotantoversiossa taas pyritään maksimoimaan käytönaikaista suorituskykyä minimoimalla Javascript- ja CSS-lähdekoodi sekä jättämällä kehitysversion lisätarkistukset

[6] `https://github.com/reworkcss/rework`
[7] `http://eslint.org/`
[8] `http://browserify.org/`

Nimi	Käyttötarkoitus
Browserify	Modulaarisen JavaScriptin kirjoittamisen mahdollistaminen.
clean-css	CSS-lähdekoodin minimointi.
ESLint	JavaScript-lähdekoodin tyylin ja virheiden tarkistus.
Rework	CSS-lähdekoodin muokkaaminen ohjelmallisesti.
Stylus	Stylus-lähdekoodin kääntäminen CSS-lähdekoodiksi.
UglifyJS 2	JavaScript-lähdekoodin minimointi.

Taulukko 5.2: Web-sovelluksen koonnissa käytettäviä työkaluja.

pois. Minimointiin käytetään clean-css[9] ja UglifyJS[10] työkaluja.

Koontivaiheessa valitaan myös sovelluksen käyttöliitymän kieli. Kaikki sovelluksen käyttöliittymän tekstit sijoitetaan erillisiin kielikohtaisiin tiedostoihin, joista ne luetaan ajonaikaisesti. Näin koontiin voidaan ottaa mukaan käännökset vain valitulle kielelle. Sovelluksen käyttöliittymä toteutetaan aluksi englanninkielisenä, mutta käännöstiedostoihin pohjautuva järjestelmä mahdollistaa uusien käännösten helpon lisäämisen.

[9] `https://github.com/jakubpawlowicz/clean-css`
[10] `https://github.com/mishoo/UglifyJS2`

Content.jsx:

```
var TabbedContent = require('./TabbedContent');
var ContentView1  = require('./ContentView1');
var ContentView2  = require('./ContentView2');

module.exports = React.createClass({
  propTypes: {
    data: React.PropTypes.object.isRequired
  },
  getInitialState: function() {
    return {
      activeTab: 0
    };
  },
  handleTabSelect: function(index) {
    this.setState({activeTab: index});
  },
  render: function() {
    var t = translations('tabbedpage');
    var data = this.props.data;
    return (
      <div className="tabcontainer">
        <TabbedContent activeTab={this.state.activeTab}
            onTabSelected={this.handleTabSelect}>
          <div title="Tab 1">
            <ContentView1 data={data}/>
          </div>
          <div title="Tab 2">
            <ContentView2 data={data}/>
          </div>
        </TabbedContent>
      </div>
    );
  }
});
```

Content.styl:

```
.tabcontainer
  background #fff
```

Listaus 5.1: Esimerkki käyttöliittymäkomponentista joka koostuu React-komponentista sekä Stylus-kielellä kirjoitetuista tyyleistä.

Luku 6

Tulokset

Tässä luvussa analysoidaan sovelluksen toteutusta eri näkökulmista ja esitetään analyysin tulokset. Aluksi esitetään havaintoja valittujen toteutustekniikoiden soveltuvuudesta hybridisovelluksen toteuttamiseen. Tämän jälkeen tarkastellaan sovelluksen osien uudelleenkäyttöä eri alustaversioissa. Lopuksi käsitellään sovelluksen sisäisiä ostoja.

6.1 Hybridisovelluksen tekniset haasteet

Seuraavissa aliluvussa esitetään havaintoja siitä, miten valitut toteutustekniikat soveltuivat monelle alustalle julkaistavan sovelluksen toteuttamiseen (tutkimuskysymys *TK1*).

6.1.1 Sovelluksen paikallisten resurssien URL-osoitteet

Cordova käyttää Web-sovelluksen paikallisten resurssien noutamiseen URL-skeemaa `file`. Tämä aiheutti ongelmia absoluuttisten URL-osoitteiden kanssa, sillä file-skeeman juuri osoittaa tiedostojärjestelmän juurihakemistoon eikä sovelluksen juurihakemistoon.

Taulukossa 6.1 on havainnollistettu ongelmaa esimerkin avulla. Tilanne A on

tyypillinen normaalissa Web-sovelluksessa, joka sijaitsee http-skeeman mukaisen URL-osoitteen juurihakemistossa. Tiedostoihin voidaan viitata jokaisessa tiedostossa samalla tavalla, sillä kaikki URL-osoitteet ovat absoluuttisia polkuja. Käytettäessä file-skeemaa sovelluksen asennushakemisto tulee osaksi sovelluksen tiedostojen URL-osoitetta, joten tämä tapa ei toimi Cordovassa.

Tilanne A. absoluuttinen polku, sovellus juurihakemistossa	
osoitteet tiedostossa `a`	osoitteet tiedostossa `h/b`
`/a` `/h/b`	`/a` `/h/b`
Tilanne B. suhteellinen polku	
osoitteet tiedostossa `a`	osoitteet tiedostossa `h/b`
`a` `h/b`	`../a` `b`
Tilanne C. absoluuttinen URL-osoite, sovellus tiedostojärjestelmän hakemistossa `/sovellus`	
osoitteet tiedostossa `a`	osoitteet tiedostossa `h/b`
`file:///sovellus/a` `file:///sovellus/h/b`	`file:///sovellus/a` `file:///sovellus/h/b`

Taulukko 6.1: Esimerkki file-skeeman aiheuttamasta ongelmasta.

Absoluuttisten polkujen sijaan on mahdollista käyttää suhteellisia polkuja (tilanne B). Tämä aiheuttaa kuitenkin omat ongelmansa. Esimerkiksi CSS-tiedostosta on hankala tehdä viittauksia ulkoisiin kuviin, kun CSS-tiedoston sijainti vaikuttaa kuvan suhteelliseen URL-osoitteeseen. Myös navigaatiolinkit HTML:ssä muuttuvat suhteellisiksi kulloinkin avoinna olevan sivun URL-osoitteeseen.

Suhteellisten polkujen käyttämisen sijaan ongelma on mahdollista ratkaista käyttämällä JavaScriptiä. Tiettyjä käyttöjärjestelmä- ja sovelluskohtaisia säännönmukaisuuksia havainnoimalla on mahdollista tunnistaa sovelluksen asennushakemisto laitteen tiedostojärjestelmässä, jolloin kunkin tiedoston absoluuttinen URL-osoite voidaan muodostaa JavaScriptiä käyttäen tie-

doston sovelluksen juurikansioon suhteellisesta polusta. Tilanne C esittää tätä lähestymistapaa käyttävän sovelluksen, joka on tunnistanut olevansa hakemistossa `/sovellus`.

Absoluuttisen URL-osoitteen muodostaminen toteutettiin sovelluksessa seuraavasti: Sovelluksen käynnistyessä luetaan avautuvan sivun URL-osoite, ja tunnistetaan siitä Cordovan käyttämä `www`-hakemisto, jossa sovellus sijaitsee. Kaikki sovelluksen käyttämät linkin muodostetaan yhdiställä `www`-hakemistoon suhteellinen polku `www`-hakemiston polkuun. Näin muodostetut osoitteet ovat absoluuttisia file-skeeman mukaisia URL-osoitteita. Sovelluksen asennushakemiston tunnistamisen JavaScript-toteutus on esitetty listauksessa 6.1.

```
if(location.protocol === 'file:') {
  var pathComponents = location.pathname.split('/');
  var baseDirIndex = pathComponents.indexOf('www');
  var baseDir = pathComponents
                .slice(0, baseDirIndex + 1)
                .join('/');
  router.setBasePath(baseDir);
}
```

Listaus 6.1: Sovelluksen asennushakemiston tunnistaminen.

6.1.2 Ulkoiset resurssit

Perinteinen tapa käyttää URL-osoitteen pelkää polkuosaa REST-kutsuissa ei toiminut hybridisovelluksessa, sillä hybridisovelluksen sisällä oleva Web-sovellus ei tiedä palvelimen osoitetta. Kaikissa REST-kutsuissa piti käyttää absoluuttista URL-osoitetta, joka sisältää myös palvelimen täydellisen osoitteen.

Kaikki kutsut hybridisovelluksesta ulkoiselle palvelimelle vaativat poikkeamista saman alkuperän käytännöstä, sillä sovellus sijaitsee tuotantoympäristöä lukuunottamatta eri alkuperässä kuin REST-rajapinta. Cordovassa tämä onnistui määrittelemällä ulkoisen palvelimen osoite erityiselle sallittujen osoitteiden listalle sovelluksen `config.xml`-tiedostoon listauksessa 6.2 esite-

tyllä tavalla. Normaalia Web-sovellusta varten tarvittiin lisäksi CORS-otsaketiedot REST-rajapinnan HTTP-vastauksiin. Listauksissa 6.3 ja 6.4 on esitetty esimerkki sovelluksen palvelimelle tekemästä HTTP-kyselystä, jossa on käytetty CORSia.

```
<access origin="http://api.trademarknow.com" />
<access origin="https://api.trademarknow.com" />
```

Listaus 6.2: Ulkoisten HTTP-pyyntöjen salliminen Cordovassa.

```
GET /api/account HTTP/1.1
Host: api.devlocal
Connection: keep-alive
Cache-Control: max-age=0
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://app.devlocal
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118
  Safari/537.36
Referer: http://app.devlocal/home
Accept-Encoding: gzip, deflate, sdch
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: sessionKey=[poistettu]
```

Listaus 6.3: HTTP-pyyntö jossa käytetään CORSia.

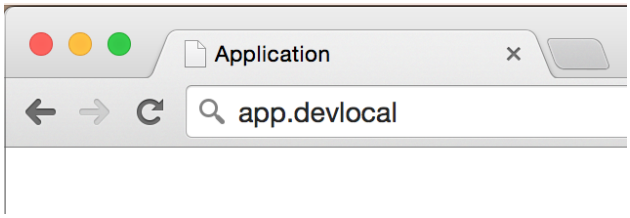
Sovelluksen ja REST-rajapinnan sijaitseminen eri alkuperissä vaikutti myös evästeiden käsittelyyn. Eri alkuperien vuoksi sovellus ei voi lukea REST-ohjelmointirajapintapalvelimen asettamia evästeitä. Palvelimen asettamat evästeet kyllä lähetetään takaisin palvelimelle HTTP-pyyntöjen mukana, mutta ne eivät näy selaimen `document.cookie`-ohjelmointirajapinnassa. Tämän takia sovelluksen on erikseen noudettava palvelimelta joitain sellaisia tietoja, jotka löytyvät asetetusta evästeestä.

```
HTTP/1.1 200 OK
Date: Tue, 21 Apr 2015 13:41:58 GMT
Server: Apache/2.4.10 (Unix)
Content-Type: application/json; charset=UTF-8
Cache-Control: private, max-age=3600
Via: 1.1 api.devlocal (Apache/2.4.10)
Access-Control-Allow-Origin: http://app.devlocal
Access-Control-Allow-Credentials: true
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
```

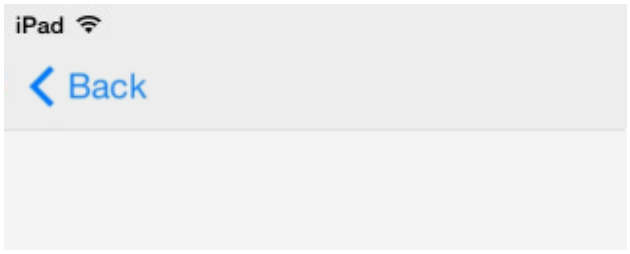
Listaus 6.4: HTTP-vastaus jossa käytetään CORSia.

6.2 Hybridisovellus ja käytettävyys

Sovelluksen käyttöliittymän osien uudelleenkäyttö eri alustaversioiden välillä (tutkimuskysymys *TK1*) johti ristiriitaan alustan käyttöliittymäohjeistuksen ja yleisten tapojen välillä. Alustalle ominaiset tavat toteuttaa jokin käyttöliittymäratkaisu erosivat paikoin merkittävästi eri alustojen välillä. Esimerkiksi edelliseen näkymään siirtyminen tapahtuu normaalissa Web-sovelluksessa Web-selaimen takaisin-painikkeella, joka sijaitsee varsinaisen sovellusalueen ulkopuolella (kuva 6.1). iOS-sovelluksissa siirtyminen tapahtuu yleensä näkymän vasemmassa ylälaidassa olevasta nuolipainikkeesta, joka on osa itse sovellusta (kuva 6.2). Sovelluksessa oleva takaisin-painike on tästä syystä piilotettu Web-alustaversiossa.



Kuva 6.1: Siirtyminen edelliseen näkymään Web-sovelluksessa.



Kuva 6.2: Siirtyminen edelliseen näkymään iOS-sovelluksessa.

Kosketusnäytöllisten laitteiden havaittiin aiheuttavan haasteita hyvän käytettävyyden saavuttamiselle. Ensinnäkin kosketusnäppäimistö tulee tarvittaessa esiin ja peitti usein osan näkyvillä olevasta sovelluksen sisällöstä. Toisekseen joissain selaimissa huomattiin olevan pieni viive kosketuksien rekisteröinnissä, sillä selain odottaa hetken mahdollista toista kosketusta, jolla käyttäjä voi suurentaa sivun sisältöä.Viiveongelma esiintyy ainakin iOS:n Safari-selaimessa ja vanhemmilla Googlen Chrome-selaimen versioilla [10]. Ongelma onnistuttiin kuitenkin kiertämään fastclick[1] nimisen JavaScript-kirjaston avulla, joka pakottaa selaimen rekisteröimään kosketukset välittömästi.

Myös mobiililaitteiden näyttöjen pienet ja vaihtelevat koot aiheuttivat haasteita käyttöliittymän sunnittelussa. Esimerkiksi pudotusvalikoiden toteutuksessa käytetty select2-kirjasto ei ole responsiivinen vaan valikon korkeus on vakio pikseleissä, mikä ei ole tehokasta, kun näyttöjen koot vaihtelevat.

6.3 Web-sovelluksen osien uudelleenkäyttö

Tässä työssä selvitettiin uudelleenkäytön onnistumista ja laajuutta hybridiversion ja Web-sovellusversion välillä (tutkimuskysymys *TK2*). Tähän käytettiin kahta mittaria: Lähdekoodista laskettiin niiden lähdekoodirivien suhteellinen määrä, jotka on tehty tiettyä alustaversiota varten. Lisäksi toteutus- ja suunnittelutyöhön käytetyistä tunneista laskettiin tiettyä alustaversiota

[1] `https://github.com/ftlabs/fastclick`

varten käytettyjen työtuntien suhteellinen määrä. Seuraavaksi esitetään tarkemmin kummankin mittarin laskentatapa ja tulokset.

Lähdekoodirivien laskentaa varten alustariippuvainen lähdekoodi sijoitettiin omiin tiedostoihinsa erilleen alustariippumattomasta lähdekoodista. Laskennassa on mielekästä tarkastella vain sovellusta varten tuotettua lähdekoodia, joten käytetyt kirjastot, sovelluskehykset ja muut valmiit osat poistettiin laskettavien tiedostojen joukosta. Tiedostojen lähdekoodirivimäärät laskettiin cloc-ohjelmalla[2], jolloin saatiin selville koko lähdekoodin yhteenlaskettu rivimäärä ja alustariippumattoman lähdekoodin yhteenlaskettu rivimäärä (taulukko 6.2). Tämän jälkeen uudelleenkäyttöaste (amount of reuse) saadaan jakamalla alustariippumattoman lähdekoodin rivimäärä koko lähdekoodin rivimäärällä. [27] Tällä tavoin koodiriveistä laskettuna uudelleenkäyttöaste eli alustariippumattoman lähdekoodin suhteellinen osuus on 94 %. Alustariippuvaisen lähdekoodin tarkastelussa selvisi, että se sisältää sovelluksen sisäiset ostot toteuttavan lähdekoodin osan.

lähdekoodin osa	**tiedostojen määrä**	**tyhjiä rivejä**	**kommentti-rivejä**	**koodi-rivejä**
alustariippumaton	30	68	7	1154
alustariippuvainen	1	17	1	63
kaikki	31	85	8	1217

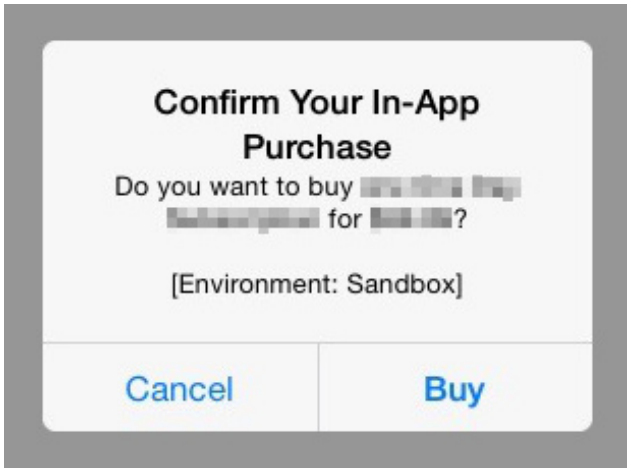
Taulukko 6.2: Alustariippumattoman- ja riippuvaisen JavaScript-lähdekoodin määrä sovelluksessa.

Työtuntien käyttöä arvioitiin seuraavasti: Aluksi kehitystyön aikana käytetyn versionhallintatyökalun historiasta määritettiin jokaiselle työviikolle tieto siitä, kohdistuiko viikon työ pääosin alustariippumattomiin vai tiettyyn alustaan kohdistuviin töihin (taulukko 6.3). Tällä tavoin laskettuna alustariippumattoman työn suhteellinen osuus on 81 %.

[2] `http://cloc.sourceforge.net/`

6.4 Sovelluksen sisäiset ostot

Sovelluksen sisäiset ostot (tutkimuskysymys *TK3*) pyrittiin toteutettamaan sovelluksen iOS-alustaversioon. Cordovaan löydettiin ainoastaan yksi laajennos, joka tarjoaa sovelluksen sisäisten ostojen toiminnallisuutta. Löydetty laajennos `cc.fovea.cordova.purchase` tarjoaa toiminnallisuuden sekä iOS- että Android-alustoille yhden ohjelmointirajapinnan kautta. Laajennos tukee Apple AppStorea ja Google Play -kauppaa. [38] Sovelluksen sisäiset ostot toteutettin onnistuneesti iOS-alustaversioon laajennoksen avulla (kuva 6.3). Koska sovellus julkaistaan muille alustoille vasta myöhemmin, sovelluksen sisäisiä ostoja ei toistaiseksi yritetty niille toteuttaa.



Kuva 6.3: Sovelluksen sisäiset ostot toteutettiin onnistuneesti sovelluksen iOS-alustaversioon.

Sovelluksen Web-alustaversiota varten tarvitaan tulevaisuudessa erillinen maksuratkaisu. Lisäosan käyttöönotto oli teknisesti kohtalaisen helppoa, vaikka sen takaisinkutsuihin perustuva ohjelmointirajapinta ei ollut kovin käytännöllinen. Myös dokumentaatio olisi voinut olla täsmällisempää varsinkin ostojen varmentamiseen liittyen.

Lisäosan käyttöönoton lisäksi sovelluksen sisäisten ostojen käyttöönotto vaati useamman sopimuksien solmimisen Applen kanssa. Sopimusta varten Apple

tarvitsi myös erinäisiä vero- ja pankkiyhteystietoja. Tämän jälkeen myytävät tuotteet luotiin sovelluskauppaan. Sopimus- ja tuotetiedot syötettiin Web-käyttöliittymän avulla.

vuosi/viikko	**työ**
2014/44	alustariippumaton
2014/45	alustariippumaton
2014/46	alustariippumaton
2014/47	alustariippumaton
2014/48	alustariippumaton
2014/49	alustariippumaton
2014/50	alustariippumaton
2014/51	alustariippumaton
2014/52	alustariippumaton
2015/01	*(loma)*
2015/02	*(loma)*
2015/03	alustariippumaton
2015/04	alustariippumaton
2015/05	alustariippumaton
2015/06	alustariippumaton
2015/07	alustariippumaton
2015/08	alustariippuvainen
2015/09	*(loma)*
2015/10	**alustariippuvainen**
2015/11	**alustariippuvainen**
2015/12	**alustariippuvainen**
2015/13	alustariippumaton
2015/14	alustariippumaton
2015/15	alustariippumaton
2015/16	alustariippumaton

Taulukko 6.3: Työviikkojen kohdistuminen alustariippumattomaan- ja riippuvaiseen työhön.

Luku 7

Johtopäätökset

Tämä luku tarkastelee työtä ja sen tuloksia laajemmassa mittakaavassa. Aluksi käsitellään hybridisovelluksen toteuttamisen vaikuttaneita tekijöitä, ja tämän jälkeen uudelleenkäytön laajuutta. Lopuksi esitetään yhteenveto työn tuloksista vastaamalla tutkimuskysymyksiin.

7.1 Web-sovelluksesta hybridisovellukseksi

Sovelluksen toteutuksen aikana tuli vastaan useita teknisiä haasteita, jotka täytyi ratkaista, jotta sovelluksen osien uudelleenkäyttö olisi mahdollista Web-sovelluksen ja hybridisovelluksen välillä, ja jotta sama Web-sovellus voitaisiin julkaista sellaisenaan hybridisovelluksena. Kuten tuloksista käy ilmi, merkittävä osa haasteista liittyi URL-osoitteisiin, jotka Cordovassa ovat file-skeeman mukaisia.

Web-sovelluskehittäjät ovat tottuneet käsittelemään http-skeeman mukaisia URL-osoitteita, joissa resurssin polku on harvoin täysin sidoksissa sen todelliseen polkuun tiedostojärjestelmässä. Yleensä sovellus sijaitsee URL-osoitteessa palvelimen juurihakemistossa, mikä helpottaa osoitteiden kanssa työskentelyä. Koska sovelluksen asennusympäristö on sovelluksen kehittäjän hallinnassa, on sovellukselle myös mahdollista kertoa sen oma sijainti. Hybridisovelluksen tapauksessa sovelluksen kehittäjä ei voi etukäteen tietää sovelluk-

sen asennushakemiston sijaintia käyttäjän laitteessa, vaan asennushakemisto on tunnistettava ajonaikaisesti.

Web-selainten ja hybridisovelluskehysten kannataisi toteuttaa minimaalinen HTTP-palvelin sovelluskehitystä ja hybridisovelluksia varten. AppGyver Steroids mainostaa tukevansa http-skeeman mukaisia URL-osoitteita, ja onkin mahdollista, että hybridisovelluskehykseksi olisi kannattanut tästä syystä valita AppGyver Steroids. Tuki http-skeeman mukaisille URL-osoitteille kannattaisi tuoda kuitenkin suoraan Cordovaan, jonka päälle Steroidskin on rakennettu. Web-selaimissa integroitu HTTP-palvelin palvelisi sovelluskehitystä, ja poistaisi file-skeemaan käyttöön liittyvät ongelmat yhtenäistämällä paikallisten ja muualla sijaitsevien resurssien noutamisen.

Koska toista alkuperää olevien evästeiden käsittely JavaScriptillä ei ole mahdollista, kannattaa hybridisovelluksissa evästeiden käyttämisen sijaan palauttaa vastaavat tiedot suoraan REST-vastauksen sisältönä. REST-rajapintaa kutsuva sovellus voi tämän jälkeen tallentaa tiedot esimerkiksi Web Storage -ohjelmointirajapintaa[1] käyttäen.

Kosketusnäytöllisten laitteiden erityispiirteet tulee ottaa huomioon hybridisovelluksen toteutuksessa huomioon samoin kuin Web-sovelluksen tapauksessa. Erityisesti joillain selaimilla oletuksena esiintyvä viive kosketusten rekisteröinnissä huonontaa käyttökokemusta suhteessa natiiveihin sovelluksiin. Myös näppäimistön esiintyöntyminen saattaa aiheuttaa ongelmia.

Uudelleenkäytettäviä käyttöliittymäkomponentteja ja interaktioita suunniteltaessa on tärkeä varmistaa, että suunnitteluratkaisut ovat luontevia jokaisen julkaisualustan kontekstissa sekä toiminnallisesti että ulkoasullisesti. Tarvittaessa on luotava erilliset ratkaisut ongelmallisille alustoille. Hybridisovellusten on havaittu olevan ongelmallisia käyttöliittymäkäytäntöjen suhteen myös aikaisemmissa tutkimuksissa [37]. Jonkinlaisen kultaisen keskitien löytäminen eri alustojen käyttöliittymäkäytännöistä vaatisi lisää tutkimusta. Käyttöliittymän tulisi tarjota hyvä käyttäjäkokemus sekä hiiri- että kosketusnäyttökäyttäjille. Myös käytettäviä käyttöliittymäkirjastoja valittaessa tulisi

[1] `http://www.w3.org/TR/webstorage/`

kiinnittää huomiota käytettävyyteen mobiililaitteilla.

Kohdatuista haasteista huolimatta voidaan todeta, että Webin alustariippumattomuus toteutui sovelluksen kehityksessä. Toteutettaessa mobiilisovelluista hybridisovelluksena on kuitenkin huomioitava mobiiliympäristön erityispiirteet ja käytettävän hybridisovelluskehyksen rajoitteet.

7.2 Sovelluksen sisäiset ostot ja uudelleenkäytön laajuus

Sovelluksen sisäisten ostojen käyttöönottoa yritettiin onnistuneesti sovelluksen iOS-alustaversiossa. Onnistumisen perusteella on oletettavaa, että sovelluksen sisäiset ostot olisi mahdollista toteuttaa myös Androidille. Sovelluksen sisäisten ostojen käytöönottoon liittyneet monimutkaisuudet liittyivät Applen AppStoreen ja olisivat tulleet myös natiivin sovelluksen kohdalla vastaan. Voidaankin todeta, että sovelluksen sisäisiä ostoja on mahdollista hyödyntää hybridisovelluksissa.

Sovelluksen lähdekoodille lasketun uudelleenkäyttöasteen perusteella voidaan sanoa, että vain pieni osa sovelluksen lähdekoodista ei toimisi sellaisenaan normaalissa Web-sovelluksessa. Alustariippuvainen lähdekoodi koostui pelkästään sovelluksen sisäiset ostot toteuttavasta osasta.

Alustariippuvaisen lähdekoodin pieneen määrään on todennäköisesti vaikuttanut se, että yksi projektin lähtökohdista oli julkaiseminen usealle alustalle. Tästä syystä toteutuksen aikana pyrittiin välttämään alustariippuvaisuuksia ja abstraktoimaan pakolliset riippuvuudet esimerkiksi käyttämällä kirjastoa. Tämän takia tulos ei ole suoraan yleistettävissä siihen kuinka hyvin satunnaisesti valittu Web-sovellus on julkaistavissa Cordovaa käyttäen hybridisovelluksena usealle alustalle. Tulosten perusteella voidaan kuitenkin todeta, että on mahdollista kehittää mobiilisovellus Web-sovelluksena niin, että se on samanaikaisesti julkaistavissa hybridisovelluksena. Tämä vaatii kuitenkin kummankin toteutustavan huomioimista projektin alusta lähtien. Mikäli alusta alkaen olisi tähdätty pelkästään tavallisen Web-sovelluksen toteuttamiseen,

olisi toteutuksesta tullut todennäköisesti monessa kohtaa yksinkertaisempi.

Webissä ei ole keskitettyä maksujärjestelmää, jota sovelluksen sisäiset ostot voisivat käyttää. Tästä syystä on odotettavaa, että sovelluksen sisäiset ostot toteuttava Cordova laajennos ei sisällä tukea Webille. Jokin Webissä tomiva maksupalvelu voisi kuitenkin lisätä laajennokseen tuen omalle maksuratkaisulleen, sillä laajennos on avointa lähdekoodia. Mikäli näin tapahtuisi, olisi koko sovelluksen lähdekoodi alustariippumatonta.

Mikäli sovellus halutaan julkaista jollekin uudelle alustalle, tarvitsee käyttöliittymää todennäköisesti kehittää jonkin verran vastaamaan uuden alustan käytäntöjä. Alustakohtaisten käyttöliittymäratkaisujen mahdollinen tarve voi johtaa alustakohtaisen lähdekoodin määrän pieneen kasvuun.

Alustariippuvaisen koodin tekemiseen ei kulunut suhteessa kovin paljon työtunteja. Työtuntien laskennan epätarkkuudesta johtuen tälle tulokselle ei kuitenkaan pidä antaa liikaa painoarvoa arvioitaessa uudelleenkäytön onnistumista.

7.3 Yhteenveto

Luvussa 1.1 esitettyihin tutkimuskysymyksiin voidaan vastata tulosten yhteenvetona seuraavasti:

TK1: Miten Web-sovellus tulee toteuttaa, jotta se on mahdollista julkaista lähes identtisenä hybridisovelluksena sovelluskaupassa?

Mikäli käytetään hybridisovelluskehystä, jossa sovelluksen paikalliset resurssit käyttävät file-skeemaa, on käytettävä suhteellisia URL-osoitteita tai muodostettava oikea absoluuttinen URL-osoite ajonaikaisesti JavaScriptin avulla. Saman alkuperän käytännön vaikutukset esimerkiksi evästeiden käsittelyyn on huomioitava käytettäessä ulkoisella palvelimella sijaitsevia resursseja. Käyttöliittymää suunniteltaessa on varmistettava, että suunnitteluratkaisut ovat luontevia jokaisen julkaisualustan kontekstissa sekä

toiminnallisesti että ulkoasullisesti.

TK2: Kuinka laajasti sovelluksen osia voidaan uudellenkäyttää hybridisovelluksen ja Web-sovelluksen välillä?

Lähes kaikkia sovelluksen osia on mahdollista uudelleenkäyttää hybridisovelluksen ja Web-sovelluksen välillä. Ainoastaan sovelluksen sisäiset ostot toteuttava Cordovan laajennos ei toimi sellaisenaan tavallisessa Web-sovelluksessa.

TK3: Onnistuuko hybridisovelluksen integroiminen yhden tai useamman sovelluskaupan sovellusten sisäisten ostojen järjestelmään?

Hybridisovellus on mahdollista integroida Applen AppStoren sovelluksen sisäisten ostojen järjestelmään käyttämällä Cordovan laajennosta. Integroiminen onnistuu samalla laajennoksella todennäköisesti myös Google Play -kauppaan.

Tämän työn tavoitteena oli kartoittaa asioita, jotka tulee ottaa huomioon, kun kehitetään hybridisovellusta mobiililaitteille. Erityisesti tarkoituksena oli löytää eroja tavallisen Web-sovelluksen toteuttamiseen. Tutkimus täytti sille asetetun tavoitteen. Tässä työssä esitetyt huomiot palvelevat hybridisovelluksien ja -sovelluskehysten kehittäjiä, sekä välillisesti hybridisovelluksena toteutettavien mobiilisovellusten käyttäjiä.

Lähteet

[1] Stylus. URL https://github.com/stylus/stylus. Viitattu 18.4.2015.

[2] Gary Anthes. HTML5 leads a Web revolution. *Communications of the ACM*, 55(7):16–17, heinäkuu 2012. ISSN 0001-0782. DOI 10.1145/2209249.2209256.

[3] Matti Anttonen, Arto Salminen, Tommi Mikkonen ja Antero Taivalsaari. Transforming the Web into a real application platform: New technologies, emerging trends and missing pieces. *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, sivut 800–807, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0113-8. DOI 10.1145/1982185.1982357.

[4] Appcelerator. Appcelerator products. URL http://www.appcelerator.com/product/. Viitattu 22.10.2014.

[5] AppGyver. AppGyver Steroids. URL http://www.appgyver.com/steroids. Viitattu 22.10.2014.

[6] Apple. In-app purchase programming guide, lokakuu 2013. URL https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/StoreKitGuide/. Viitattu 31.3.2015.

[7] Apple. About the iOS technologies, syyskuu 2014. URL https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/. Viitattu 4.3.2015.

[8] Apple. Start developing iOS apps today, joulukuu 2014. URL `https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/`. Viitattu 4.3.2015.

[9] Apple. iOS developer program. URL `https://developer.apple.com/programs/ios/`. Viitattu 4.3.2015.

[10] Jake Archibald. 300ms tap delay, gone away, joulukuu 2013. URL `http://updates.html5rocks.com/2013/12/300ms-tap-delay-gone-away`. Viitattu 24.3.2015.

[11] Lee Barney. QuickConnect iPhone: an iPhone UIWebView hybrid framework, toukokuu 2008. URL `https://tetontech.wordpress.com/2008/05/28/quickconnect-iphone-an-iphone-hybrid-framework/`. Viitattu 30.1.2015.

[12] Lee Barney. UIWebView example code, toukokuu 2008. URL `https://tetontech.wordpress.com/2008/05/23/uiwebview-example-code/`. Viitattu 30.1.2015.

[13] Adam Barth. The Web origin concept. RFC 6454 (Proposed Standard), joulukuu 2011. URL `http://www.ietf.org/rfc/rfc6454.txt`.

[14] Robin Berjon, Steve Faulkner, Ian Hickson, Travis Leithead, Erika Doyle Navara, Edward O'Connor ja Silvia Pfeiffer. HTML5. W3C Recommendation, W3C, lokakuu 2014. URL `http://www.w3.org/TR/2014/REC-html5-20141028/`.

[15] Tim Berners-Lee, Roy Fielding ja Larry Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986 (Internet Standard), tammikuu 2005. URL `http://www.ietf.org/rfc/rfc3986.txt`.

[16] Blackberry. Develop with BlackBerry. URL `http://developer.blackberry.com/develop/`. Viitattu 31.3.2015.

[17] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159 (Proposed Standard), maaliskuu 2014. URL `http://www.ietf.org/rfc/rfc7159.txt`.

[18] Tim Bray, Jean Paoli, Michael Sperberg-McQueen, Eve Maler ja François Yergeau. Extensible markup language (XML) 1.0 (fifth edition). W3C Recommendation, W3C, marraskuu 2008. URL `http://www.w3.org/TR/2008/REC-xml-20081126/`.

[19] Laurie M. Bridges ja Hannah Gashco Rempel. That was then, this is now: Replacing the mobile-optimized site with responsive design. *Information Technology and Libraries*, 32(4):8–24, 2013. DOI `10.6017/ital.v32i4.4636`.

[20] Andre Charland ja Brian Leroux. Mobile application development: Web vs. native. *Communications of the ACM*, 54(5):49–53, toukokuu 2011. ISSN 0001-0782. DOI `10.1145/1941487.1941504`.

[21] Isabelle Dalmasso, Soumya K. Datta, Christian Bonnet ja Navid Nikaein. Survey, comparison and evaluation of cross platform mobile application development tools. *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, sivut 323–328, heinäkuu 2013. DOI `10.1109/IWCMC.2013.6583580`.

[22] Facebook. Advanced performance. URL `https://facebook.github.io/react/docs/advanced-performance.html`. Viitattu 18.4.2015.

[23] Facebook. JSX in depth. URL `https://facebook.github.io/react/docs/jsx-in-depth.html`. Viitattu 18.4.2015.

[24] Roy T. Fielding ja Julian F. Reschke. Hypertext transfer protocol (HTTP/1.1): Message syntax and routing. RFC 7230 (Proposed Standard), kesäkuu 2014. URL `http://www.ietf.org/rfc/rfc7230.txt`.

[25] Roy T. Fielding ja Richard N. Taylor. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, toukokuu 2002. ISSN 1533-5399. DOI `10.1145/514183.514185`.

[26] Apache Software Foundation. About Apache Cordova. URL `https://cordova.apache.org/`. Viitattu 22.10.2014.

[27] William Frakes ja Carol Terry. Software reuse: Metrics and models. *ACM Comput. Surv.*, 28(2):415–435, kesäkuu 1996. ISSN 0360-0300. DOI 10.1145/234528.234531.

[28] Jesse James Garrett. Ajax: A new approach to Web applications, helmikuu 2005. URL http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. Viitattu 19.2.2015.

[29] Gartner. Gartner says annual smartphone sales surpassed sales of feature phones for the first time in 2013, helmikuu 2014. URL http://www.gartner.com/newsroom/id/2665715. Viitattu 14.10.2014.

[30] Gartner. Gartner says worldwide tablet sales grew 68 percent in 2013, with android capturing 62 percent of the market, maaliskuu 2014. URL http://www.gartner.com/newsroom/id/2674215. Viitattu 14.10.2014.

[31] Gartner. Gartner says sales of tablets will represent less than 10 percent of all devices in 2014, lokakuu 2014. URL http://www.gartner.com/newsroom/id/2875017. Viitattu 14.1.2015.

[32] Gartner. Gartner says in 2015, 50 percent of people considering buying a smart wristband will choose a smartwatch instead, marraskuu 2014. URL http://www.gartner.com/newsroom/id/2913318. Viitattu 14.1.2015.

[33] Google. Android SDK. URL https://developer.android.com/sdk/. Viitattu 23.10.2014.

[34] Google. The Android source code. URL http://source.android.com/source/. Viitattu 14.10.2014.

[35] Google. Google Play in-app billing. URL https://developer.android.com/google/play/billing/index.html. Viitattu 31.3.2015.

[36] John Gregg ja Anne van Kesteren. Web notifications. W3C Recommendation, W3C, syyskuu 2013. URL http://www.w3.org/TR/2013/WD-notifications-20130912/.

[37] Henning Heitkötter, Sebastian Hanschke ja Tim A. Majchrzak. Evaluating cross-platform development approaches for mobile applications. Teoksessa *Web Information Systems and Technologies*, José Cordeiro ja Karl-Heinz Krempels, toimittajat, osa 140 sarjasta *Lecture Notes in Business Information Processing*, sivut 120–138. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36607-9. DOI 10.1007/978-3-642-36608-6_8.

[38] Jean-Christophe Hoelt. Cordova purchase plugin, lokakuu 2014. URL https://github.com/j3k0/cordova-plugin-purchase. Viitattu 18.4.2015.

[39] Adrian Holzer ja Jan Ondrus. Mobile application market: A developer's perspective. *Telematics and Informatics*, 28(1):22–31, 2011. ISSN 0736-5853. DOI 10.1016/j.tele.2010.05.006. Mobile Service Architecture and Middleware.

[40] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion ja Steven B. Byrne. Document object model (DOM) level 3 core specification. W3C Recommendation, W3C, huhtikuu 2004. URL http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407.

[41] Sami Hyrynsalmi, Tuomas Mäkilä, Antero Järvi, Arho Suominen, Marko Seppänen ja Timo Knuutila. App store, marketplace, play! an analysis of multi-homing in mobile software ecosystems. *Proceedings of the International Workshop on Software Ecosystems*, sivut 59–72, 2012.

[42] Sami Hyrynsalmi, Arho Suominen, Tuomas Mäkilä, Antero Järvi ja Timo Knuutila. Revenue models of application developers in Android Market ecosystem. Teoksessa *Software Business*, MichaelA. Cusumano, Bala Iyer ja N. Venkatraman, toimittajat, osa 114 sarjasta *Lecture Notes in Business Information Processing*, sivut 209–222. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30745-4. DOI 10.1007/978-3-642-30746-1_17.

[43] Slinger Jansen ja Ewoud Bloemendal. Defining app stores: The role of curated marketplaces in software ecosystems. Teoksessa *Software Business. From Physical Products to Software Services and Solutions*, Georg Herzwurm ja Tiziana Margaria, toimittajat, osa 150 sarjasta *Lecture Notes in Business Information Processing*, sivut 195–206. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39335-8. DOI 10.1007/978-3-642-39336-5_19.

[44] David Jaramillo, Robert Smart, Borko Furht ja Ankur Agarwal. A secure extensible container for hybrid mobile applications. *Southeastcon, 2013 Proceedings of IEEE*, sivut 1–5, huhtikuu 2013. DOI 10.1109/SECON.2013.6567439.

[45] Mona E. Joorabchi, Ali Mesbah ja Philippe Kruchten. Real challenges in mobile app development. *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, sivut 15–24, lokakuu 2013. DOI 10.1109/ESEM.2013.9.

[46] Antero Juntunen, Eetu Jalonen ja Sakari Luukkainen. HTML 5 in mobile devices – drivers and restraints. *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, sivut 1053–1062, tammikuu 2013. DOI 10.1109/HICSS.2013.253.

[47] Ethan Marcotte. Responsive Web design, toukokuu 2010. URL http://alistapart.com/article/responsive-web-design. Viitattu 1.4.2015.

[48] Ali Mesbah ja Arie van Deursen. Migrating multi-page web applications to single-page AJAX interfaces. *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, sivut 181–190, maaliskuu 2007. DOI 10.1109/CSMR.2007.33.

[49] Microsoft. Account types, locations, and fees. URL https://msdn.microsoft.com/en-us/library/windows/apps/jj863494.aspx. Viitattu 31.3.2015.

[50] Microsoft. How to develop a store app. URL https://msdn.microsoft.com/library/windows/apps/xaml/dn726537.aspx. Viitattu 9.2.2015.

[51] Sanja Mohorovičić. Implementing responsive web design for enhanced web presence. *Information Communication Technology Electronics Microelectronics (MIPRO), 2013 36th International Convention on*, sivut 1206–1210, toukokuu 2013.

[52] Jakob Nielsen. Mobile site vs. full site, huhtikuu 2012. URL `http://www.nngroup.com/articles/mobile-site-vs-full-site/`. Viitattu 1.4.2015.

[53] OpenSignal. Android fragmentation visualized, elokuu 2012. URL `http://opensignal.com/reports/fragmentation.php`. Viitattu 21.10.2014.

[54] Alex Russell, Jungkee Song ja Jake Archibald. Service workers. W3C Recommendation, W3C, helmikuu 2015. URL `http://www.w3.org/TR/2015/WD-service-workers-20150205/`.

[55] StatCounter. StatCounter global stats comparison from Mar 2014 to Feb 2015. URL `http://gs.statcounter.com/#all-comparison-ww-monthly-201403-201502`. Viitattu 1.4.2015.

[56] Adobe Systems. PhoneGap developer portal, 2015. URL `http://phonegap.com/developer/`. Viitattu 18.4.2015.

[57] A. Taivalsaari ja Kari Systä. Cloudberry: An HTML5 cloud phone platform for mobile devices. *Software, IEEE*, 29(4):40–45, heinäkuu 2012. ISSN 0740-7459. DOI `10.1109/MS.2012.51`.

[58] Antero Taivalsaari ja Tommi Mikkonen. The Web as an application platform: The saga continues. *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, sivut 170–174, elokuu 2011. DOI `10.1109/SEAA.2011.35`.

[59] Anne van Kesteren. Cross-origin resource sharing. W3C Recommendation, W3C, tammikuu 2014. URL `http://www.w3.org/TR/2014/REC-cors-20140116/`.

[60] Anne van Kesteren, Julian Aubourg, Jungkee Song ja Hallvord Steen. XMLHttpRequest level 1. W3C Recommenda-

tion, W3C, tammikuu 2014. URL http://www.w3.org/TR/2014/WD-XMLHttpRequest-20140130/.

[61] Anthony I. Wasserman. Software engineering issues for mobile application development. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, sivut 397–400, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0427-6. DOI 10.1145/1882362.1882443.

[62] Joel West ja Michael Mace. Browsing as the killer app: Explaining the rapid success of Apple's iPhone. *Telecommunications Policy*, 34(5–6): 270–286, 2010. ISSN 0308-5961. DOI 10.1016/j.telpol.2009.12.002.